WL-TR-93-3005

A REAL-TIME, HARDWARE-IN-THE-LOOP SIMULATION OF AN UNMANNED AERIAL RESEARCH VEHICLE

SCOTT D. ROBERTSON
FLIGHT CONTROL DIVISION
FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY (AFMC)

AUG 1992

FINAL REPORT FOR 05/01/91-08/28/92

FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7521

[illegible]	[illegible]
SCOTT D. ROBERTSON, Capt, USAF Project Engineer Control Systems Development & Applications Branch Flight Control Division	RUDY C. BEAVIN Control Data Group Section Leader Control Systems Development & Applications Branch Flight Control Division

[illegible]

DAVID P. LEMASTER
Chief, Flight Control Division

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE AUG 1992	3. REPORT TYPE AND DATES COVERED FINAL 05/01/91--08/28/92

4. TITLE AND SUBTITLE
A REAL-TIME, HARDWARE-IN-THE-LOOP SIMULATION OF AN UNMANNED AERIAL RESEARCH VEHICLE

5. FUNDING NUMBERS
C
PE 62201
PR 2403
TA 07
WU 50

6. AUTHOR(S)
SCOTT D. ROBERTSON

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
FLIGHT CONTROL DIVISION
FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY (AFMC)

8. PERFORMING ORGANIZATION REPORT NUMBER
WL-TR-93-3005

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
FLIGHT DYNAMICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WL/FIGS, Attn: ROBERTSON 513-2558290

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
WL-TR-93-3005

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Simulation is a valuable tool for rapidly and cost effectively developing and verifying new systems. This report describes the design and development of a system which simulates an unmanned aircraft and verifies the proper operation of its flight computer. Flight computers, being time critical devices, must execute at a certain rate in order to safely and accurately control an aircraft. In order to verify the proper operation of the flight computer, the simulation must execute in 'real-time', which is a rate of 100 Hz for this flight computer. The flight computer's inputs and outputs are connected to the simulation so that the flight computer thinks it is flying.

The simulation architecture is a VME-based computer system, consisting of an input board (designed for this project) which demodulates 13 channels of pulse-width modulated signals, a processor board which executes the simulation software, and a digital-to-analog output board. The existing simulation software was ported to the C language and re-hosted on a 68030 processor board. Additional software was written to handle all the I/O and data conversions. The system accurately simulates the unmanned aircraft, runs in real-time, and verifies the proper operation of the flight computer.

14. SUBJECT TERMS
Unmanned, Unmanned Aerial Vehicle, UAV, Simulation, Hardware in the Loop simulation VME-based system

15. NUMBER OF PAGES
64

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

CONTENTS

FIGURES

SECTION 1 - INTRODUCTION

1.1 BACKGROUND

The Flight Dynamics Directorate (FIGL), Wright Laboratory, Wright-Patterson Air Force Base, utilizes an Unmanned Research Vehicle (URV) to conduct experiments in flight control and aerospace vehicle management (Figure 1). By using an unmanned vehicle, FIGL can flight test new control algorithms, flight computer architectures, and sensor technology that would be too risky and/or expensive for manned flight tests. Once these technologies have been tested and refined using unmanned flight tests, they can be transitioned to manned aircraft with significantly lower risk.

To reduce the risks to both the ground crew and the vehicle itself, there is a need for a ground based simulation which can thoroughly exercise the on-board flight computer. Such a capability would allow new hardware and software technologies to be tested and debugged on the ground before they are flown. This would significantly reduce the inherent risks of flight testing new experiments, decrease the debugging time for both developing and integrating experiments into the vehicle, and increase the chances for successful flight tests.

This thesis project covers the design, development, and test of a real time, hardware-in-the-loop simulation of an unmanned aerial research vehicle. Since it is the flight computer, the brain which controls the URV, that is constantly changing and being tested during flight control algorithm and computer architecture experiments, it is imperative that it operates correctly before being flown. Prior to this project, only limited testing could be performed on the flight computer. The purpose of this project is to provide the capability to verify the proper operation of the aircraft's flight computer prior to flight testing. This will allow the pilot to practice "flying" the flight computer and get a feel for how the aircraft will respond to the control algorithms or computer architecture being tested. It will also allow aircraft state and control data to be collected for comparison against how the aircraft would normally respond and against how the new algorithm or architecture was designed to respond. This capability will allow experiments to be rapidly developed and debugged since they will not have to go through iterative flight tests for initial data collection, and will greatly reduce the risks involved in flying new flight control experiments.

1.2 STATEMENT OF PROBLEM

The objective of this thesis project is to design and develop an architecture for a real time simulation of an unmanned aerial research vehicle. The simulation must be capable of non-intrusively verifying proper operation of the vehicle's on-board flight computer. This includes the design and development of a pulse-width demodulation circuit board to decode the flight computer commands sent to the aircraft control surfaces. Real time response of the simulation is critical for verifying proper operation

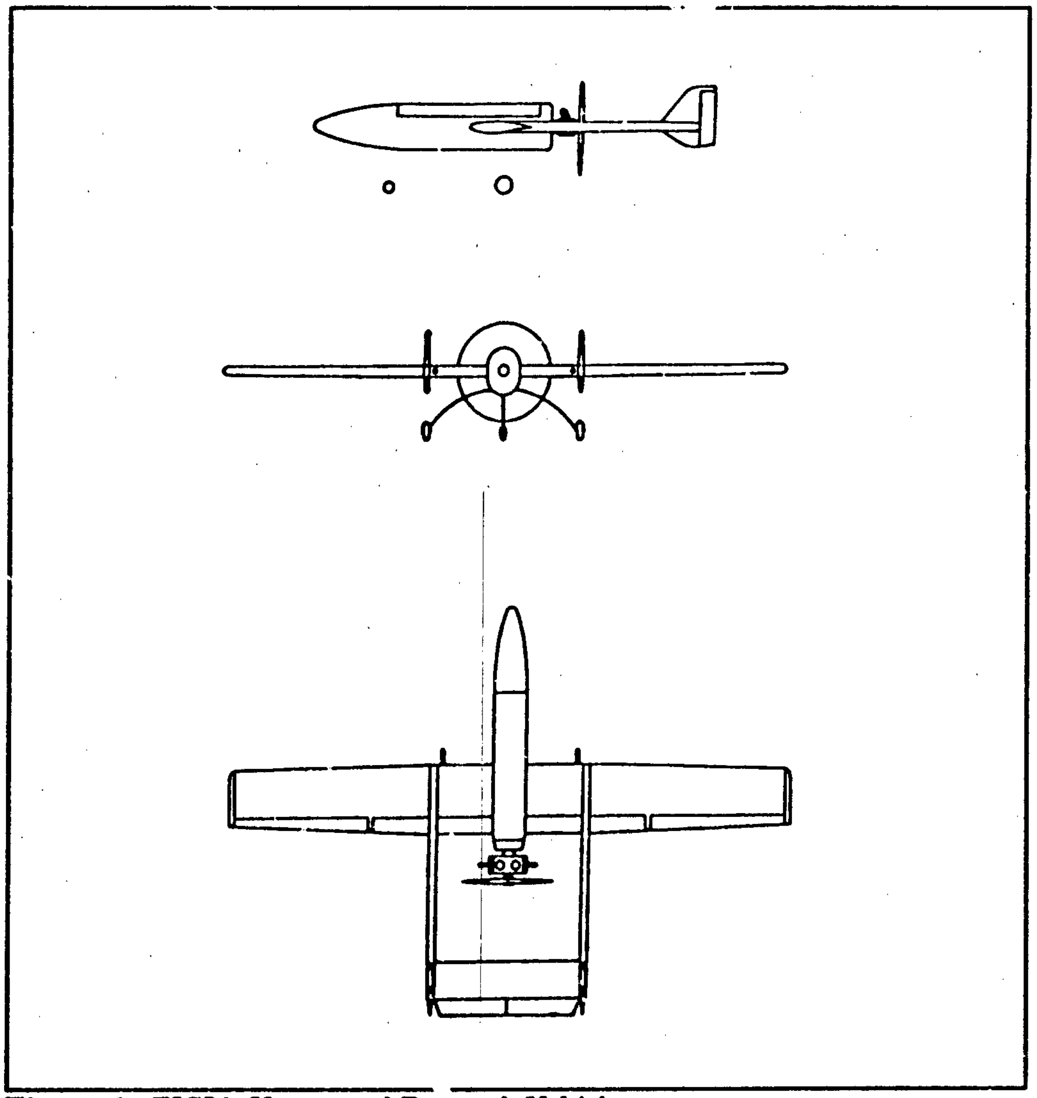


Figure 1: FIGL's Unmanned Research Vehicle

of experimental flight control algorithms and hardware configurations in the flight computer. The simulation must be able to fully exercise the flight computer such that it will respond as if it were in-flight. This will provide the capability to perform laboratory tests with real time response on actual flight hardware and software, verifying proper operation of hardware and/or software experiments prior to flight testing.

1.3 OVERVIEW OF DESIGN REQUIREMENTS

It is theoretically possible to connect an aircraft flight computer to a simulation of the aircraft and "fly" the flight computer. To do this, the simulation must process the flight computer outputs, compute the next state of the vehicle (its roll, pitch, and yaw rates and attitudes, airspeed, altitude, and side-slip), and generate the necessary flight computer input signals. The flight computer can then be "flown", behaving as if it was flying in the aircraft. There are four main elements which are required to accomplish this type of "hardware-in-the-loop" simulation: an accurate computer model of the aircraft, the computing power to execute the model in real time, the interfaces for receiving and decoding the flight computer outputs, and the interfaces for generating the flight computer inputs.

The configuration for the most accurate hardware-in-the-loop simulation utilizes the flight computer in a non-invasive manner. That is, the flight computer is taken as a whole and connected to the simulation in the same manner that it is connected to the aircraft. This tests the input and output functions of the flight computer as well as the control algorithms, producing the highest degree of confidence that it will perform the same while controlling the aircraft in the air as it does in the simulation. This thesis project will use this type of non-invasive interface to the flight computer.

The flight computer uses pulse-width modulated outputs to command the vehicle's control surfaces (ailerons, elevators, and rudders). A pulse-width signal is a digital signal (either high or low) with a fixed period, who's high-time (the width of the pulse) varies. The simulation system must take these pulse width signals, decode them into appropriate values, and provide them for input to the simulation software. Based on these inputs and the previous states of the aircraft, the simulation calculates the next state of the aircraft. The "next state of the aircraft" is the orientation and rates that the aircraft should be in at the start of the next execution of the flight computer control code. This "future" state of the aircraft must be converted into the appropriate sensor signals which the flight computer uses as inputs for controlling the vehicle. The flight computer executes its flight control code iteratively at a rate of 50 Hz. For real time operation, the simulation must guarantee that it can decode all necessary channels of pulse-width signals, generate the next state of the aircraft, and provide the simulated sensor signals back to the flight computer before the flight computer starts its next flight control execution cycle.

1.4 EXISTING SIMULATION CAPABILITIES

There are some specialized facilities available which can perform accurate aircraft simulations in real time and provide simulated sensor feedback to the flight computer. However, these facilities do not have the capability to decode all the necessary pulse-width signals generated by this flight computer. Instead, they must invade the flight computer and get the control surface commands before they are converted to pulse-width signals. This slightly lowers the degree of confidence that the flight computer will operate the same while controlling the vehicle in flight. These facilities are expensive and require a 1 to 2 week lead time to schedule the facility and set up the experiment. In contrast, the development costs for this project are about equal to four simulation runs at such facilities, and incur no additional cost per simulation. Simulation turn-around times will be almost immediate, providing much higher utiliza[illegible] of manpower and facilities, and shorter lead times for setting up and executing flig[illegible] periments.

1.5 SEQUENCE OF PRESENTATION

Section 2 analyzes the timing requirements for achieving a real time simulation as well as the signal types required for interfacing to the flight computer. The hardware elements for the simulation (decode pulse-width signal, execute simulation software, generate simulated sensor signals) are defined in Section 3. Section 4 explains the design of a pulse-width demodulation circuit board and its operation. The software for the operation of the pulse-width demodulation board and the software for the aircraft simulation are described in Section 5. The test and verification of the simulation is presented and conclusions are drawn in Section 6.

SECTION 2 - SIMULATION SYSTEM REQUIREMENTS

2.1 TIMING REQUIREMENTS

The flight computer executes its software iteratively at a rate of 50 Hz. This means that once every 20 milliseconds (ms) the flight computer samples the state of the aircraft from the aircraft sensors, processes this information with the pilot commands, and generates commands for each control surface (ailerons, elevators, rudders). Aircraft sensors provide rate, attitude, altitude, airspeed, and side-slip information.

In order to provide real time response, the simulation must decode the control surface command signals, calculate the next state of the vehicle, and output simulated sensor signals (which reflect the new state of the vehicle) at least once every 20 ms. See Figure 2. This is the only way to guarantee an accurate response from the flight computer.

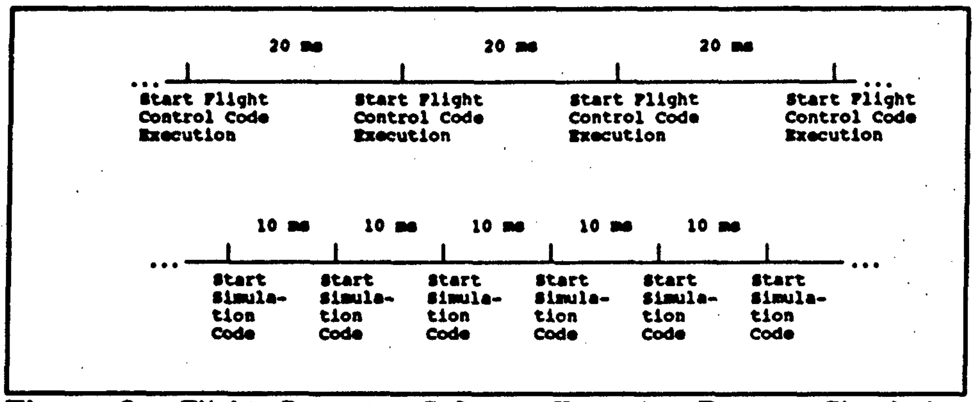


Figure 2: Flight Computer Software Execution Rate vs Simulation Software Execution Rate

The only signals being generated by the flight computer are the pulse-width commands. Though the pulse-width commands are updated by the flight computer once every 20 ms, the pulse-widths are continuously being generated in order to maintain a control surface position. Since the simulation is totally non-invasive to the flight computer and there are no timing signals generated by the flight computer, there is no way to synchronize the execution of the simulation software with the execution of the flight computer software. In order to guarantee that the simulation samples the pulse-width commands and generates the next state of the aircraft within any given 20 ms window (the execution rate of the flight computer software), the simulation software must execute at a minimum of twice the rate of the flight computer software. This means that the simulation software must execute at least once every 10 ms (100 Hz). If it is any slower, it will periodically be too late in providing the next state of the aircraft to the flight computer. For example, if the simulation software executes once every 11 ms, then the situation will occur when the simulation software starts executing 1 ms before the flight computer begins executing, as is shown in execution frame F_{n+1} in Figure 3. The simulation will sample once during frame F_{n+1}. However, it will not provide the state of the aircraft data until frame F_{n+2}, which is too late. The flight computer will have already sampled the state of the aircraft before the simulation has provided it for frame F_{n+2}. Accurate flight computer response depends on the simulation sampling the surface positions and updating the sensor signals within every frame of the flight computer software execution.

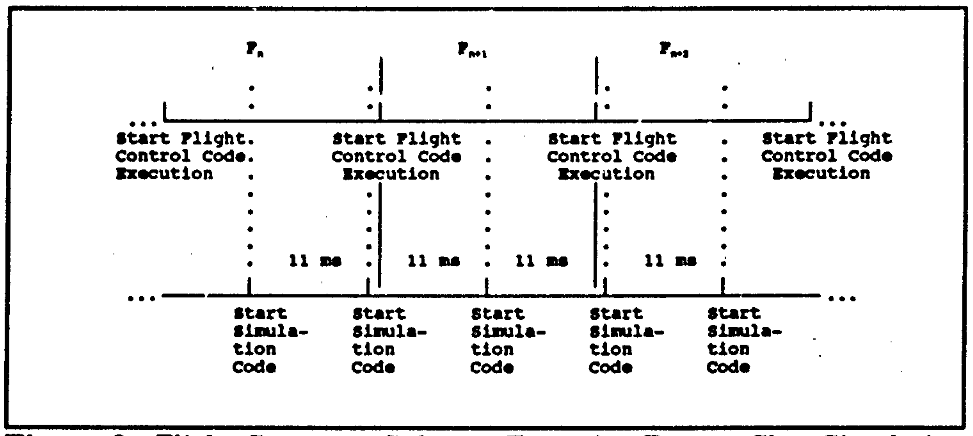


Figure 3: Flight Computer Software Execution Rate vs Slow Simulation Software Execution Rate

In the worst case, if the flight computer is executing a very sensitive controller and the simulation takes longer than half the execution rate of the flight computer software, the flight computer will occasionally sample old sensor data (the same data it sampled the previous time it executed). The flight computer, thinking that the vehicle is not responding, will increase the magnitude of its control surface commands. When the simulation processes the larger commands and feeds the new state of the vehicle back to the flight computer, the flight computer will realize that the vehicle has over responded and will reduce the control surface commands to compensate. This can result in an oscillatory vehicle response, producing no useful simulation data. In fact, it would appear that the flight computer cannot adequately control the vehicle, when it is actually a simulation problem.

2.2 SIMULATION SOFTWARE

The software model of the air vehicle was previously developed on a Sun workstation using a simulation package called Easy V. A stand-alone Fortran program was provided for this simulation project. The model requires extensive double precision calculations and trigonometric functions.

2.3 SIGNAL REQUIREMENTS

2.3.1 Flight Computer Output Signals

The flight computer outputs 13 channels of TTL level (0-5 volts) pulse-width modulated signals to drive the actuators, which in turn control the position of the aircraft control surfaces. The channels are as follows (see Figure 4 for a top view of the aircraft's control surfaces):

2 aileron channels (left and right)

2 elevator channels (left and right)

2 rudder channels (left and right)

4 flap channels (left and right inboard and outboard)

1 throttle channel

1 steering channel

1 brake channel

Though the simulation samples all 13 channels to verify that they are operating properly, only 3 channels are required for the simulation software: one aileron channel, one elevator channel, and one rudder channel. The same pulse-width command is given for right and left channels, so only one of each needs to be sampled.

The pulse-widths are generated continuously with a period of 2 ms. The high time varies between 1 ms and 2 ms. A 1 ms pulse width generates a full control surface deflection in one direction and a 2 ms pulse width generates a full control surface deflection in the opposite direction. See Figure 5. A pulse width of 1.5 ms places the control surface in the neutral position. Though the flight computer commands 12-bits of accuracy for the pulse-widths (a commanded resolution of 244 nanoseconds (ns) (1 ms/2^{12} = 244 ns)), it can only resolve its pulse-width outputs to 2 microseconds. In order to verify the accuracy of the pulse-width commands, the simulation must decode these signals to the 2 microsecond (μs) resolution.

2.3.2 Sensor Signals

The flight computer requires sensor information to control the vehicle. This

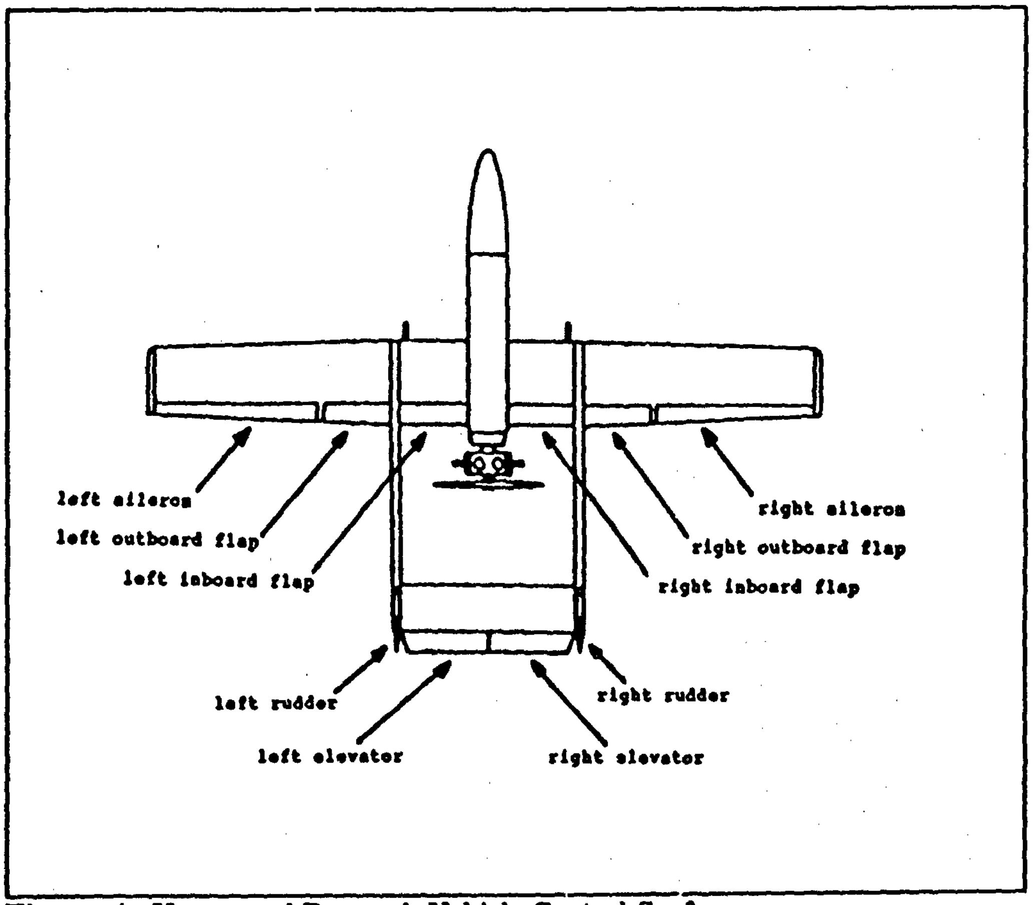


Figure 4: Unmanned Research Vehicle Control Surfaces

information arrives in analog form from each sensor. Table 1 shows each sensor channel, the type of signal it outputs, and the range of values it represents. The flight computer has a 12-bit analog-to-digital converter which it uses to acquire this information. The simulation, in turn, should continuously provide analog information to the flight computer that is accurate to 12 bits for each of the channels in Table 1.

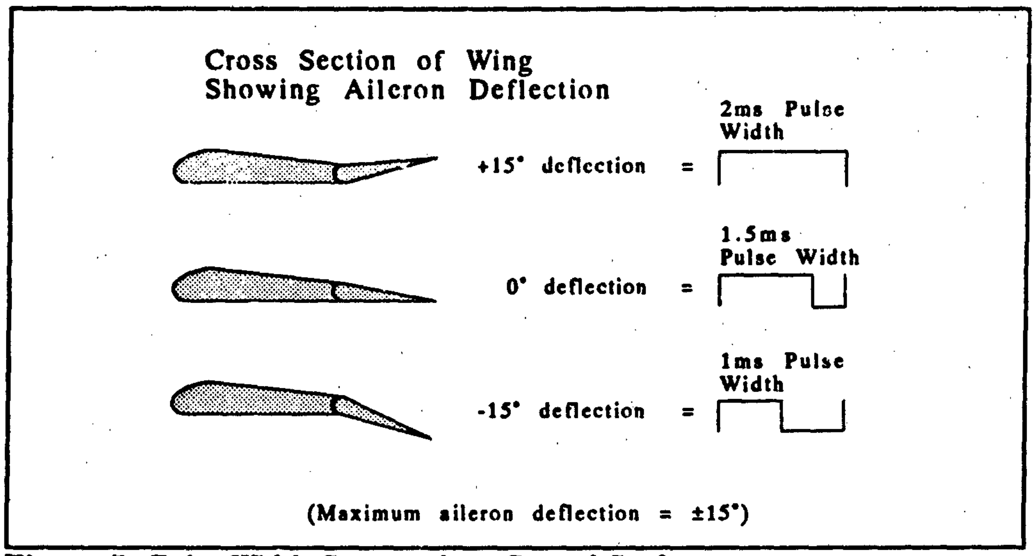


Figure 5: Pulse-Width Commands to Control Surfaces

Table 1: Air Vehicle Sensor Channels

Sensor Channel	Signal Type	Range
Roll rate	0 - 5 volts analog	-50 to 50 deg/sec
Pitch rate	0 - 5 volts analog	-50 to 50 deg/sec
Yaw rate	0 - 5 volts analog	-50 to 50 deg/sec
Roll attitude	0 - 5 volts analog	-90 to 90 degrees
Pitch attitude	0 - 5 volts analog	-60 to 60 degrees
Yaw attitude	0 - 5 volts analog	-90 to 90 degrees
Altitude	0 - 5 volts analog	0 to 8000 feet
Air speed	0 - 5 volts analog	0 to 150 knots

SECTION 3 - SYSTEM HARDWARE DESIGN

3.1 HARDWARE DESIGN OVERVIEW

The aircraft's flight computer, with its embedded control algorithms, will be connected to a system which simulates the air vehicle. Figure 6 shows the top level design approach for how the flight computer will be connected to the simulation. The simulation system must be capable of decoding the flight computer outputs, executing the vehicle simulation code which calculates the next state of the air vehicle, and generating simulated sensor signals for input to the flight computer. After considering a number of implementation alternatives, the choice was narrowed down to a personal computer-based simulation or a VMEbus-based solution. The VMEbus, having a number of important advantages over a personal computer-based system, was chosen. The advantages include the availability of high performance I/O boards and processor boards, ease of upgrading to higher performance processors and additional I/O, and the ability to have multiple processor configurations for low cost, high performance systems. Since the flight control computer is VME based, there are added benefits to choosing a VME-based simulation system. Existing VME development tools for the flight computer can be used in the development of the simulation system. Also, as one of the steps in developing the simulation system, the simulation software can be embedded into the flight computer at an early stage to verify that the simulation software provides an accurate simulation of the air vehicle and can execute in real time.

3.2 HARDWARE MODULES

A market survey of VME circuit boards showed that in order to meet the functional requirements of the simulation, three separate boards are required (as shown in Figure 7). A digital-to-analog (D/A) board is required to generate the 0-5 volt analog simulated sensor signals, and a high performance processor card with a math coprocessor is required to execute the simulation code. However, there are currently no VME boards on the market which can convert thirteen channels of pulse-width modulated signals to digital values. Some boards can convert two channels of pulse-width signals, but using seven such boards to convert 13 channels is not cost-effective and will not fit in a standard eight slot VME rack (two slots are already required for the D/A board and processor board). Therefore, a pulse-width demodulation board had to be designed and built in order to run the simulation.

3.2.1 Processor Board

The Heurikon HK68/V3D VMEbus 68030-based Single Board Computer was used to host the simulation software. This processor board uses a Motorola 68030 microprocessor running at 32 MHz and a Motorola 68882 floating point coprocessor.

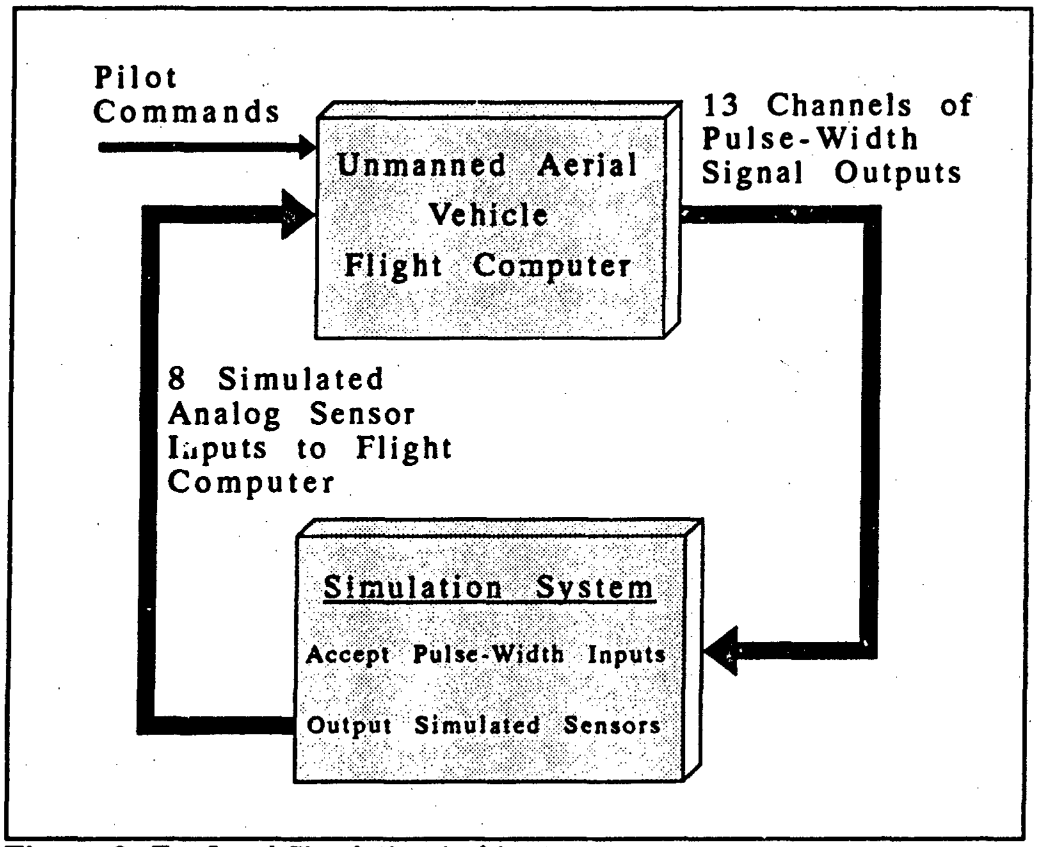

Pilot Commands
Unmanned Aerial Vehicle Flight Computer
13 Channels of Pulse-Width Signal Outputs
8 Simulated Analog Sensor Inputs to Flight Computer
Simulation System
Accept Pulse-Width Inputs
Output Simulated Sensors

Figure 6: Top Level Simulation Architecture

It has 12 megabytes of RAM (random access memory) and an EPROM (erasable/programmable read-only memory) capacity of 2 megabytes. The board supports full VMEbus addressing, bus mastering, and interrupt capabilities. Two serial I/O ports and an RS-232C interface are provided. The board also has a Zilog Z8536 counter/timer and parallel I/O unit.

3.2.2 Digital-to-Analog Board

The VME Microsystems International VMIVME-4132 32-Channel 12-Bit Analog Output Board was chosen to provide the required analog signals. This is the only 32-channel 12-bit D/A VME board on the market and was chosen to provide the simulation system with extra channels of D/A for future expandability. This board provides 10mA (milliampere) of drive current over the full output range of -10 to +10 volts. The output ranges can be set as 0 to 5 V, 0 to 10 V, -2.5 to +2.5 V, -5 to +5 V, and -10 to +10 V. Output accuracy is 0.005 percent, and all outputs are updated every 3.4 ms. The output refresh rate can be increased to update all channels every 0.85 ms with slightly reduced output accuracy. The VMIVME-4132 also features a

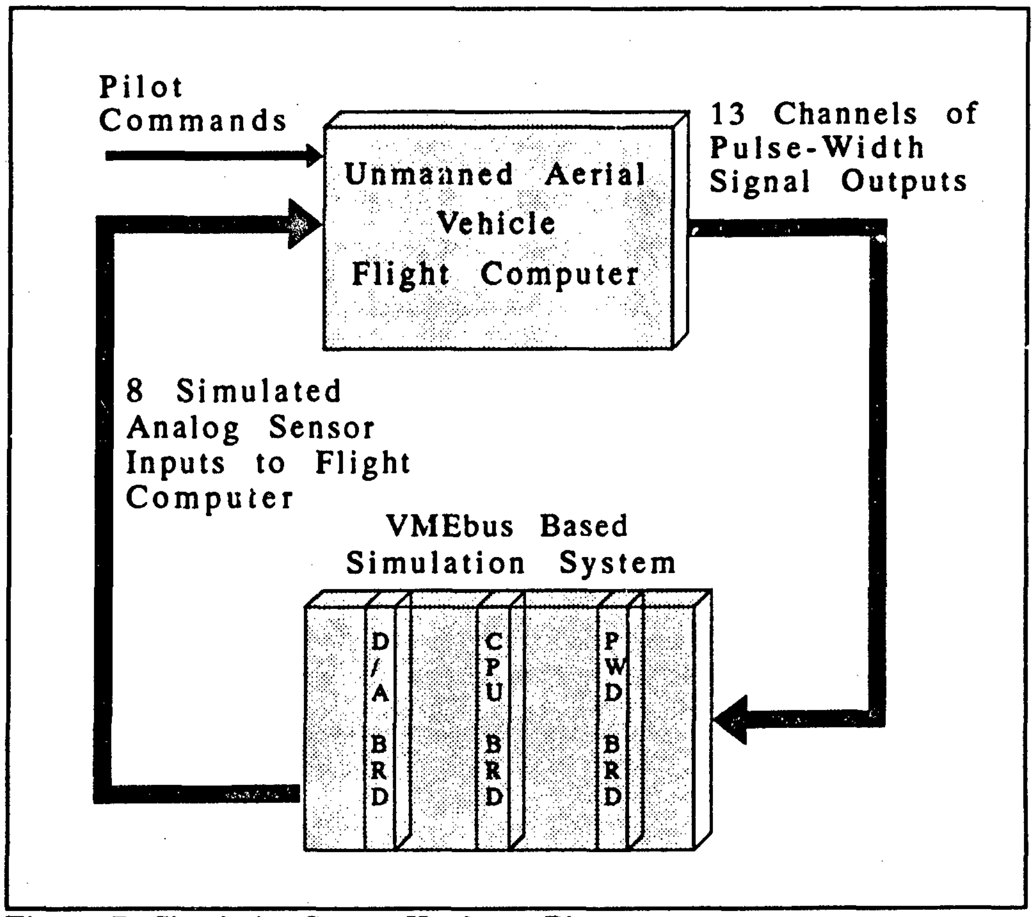

Pilot Commands
Unmanned Aerial Vehicle Flight Computer
13 Channels of Pulse-Width Signal Outputs
8 Simulated Analog Sensor Inputs to Flight Computer
VMEbus Based Simulation System
D/A BRD
CPU BRD
PWD BRD

Figure 7: Simulation System Hardware Diagram

loopback test capability to measure the voltage outputs of all channels.

3.2.3 Pulse-Width Demodulation

As mentioned, there are no pulse-width demodulation boards on the market which support the required 13 channels of pulse-width demodulation. A market survey produced only a couple of microcontrollers with multi-channel pulse-width demodulation capability. The Motorola MC68332 microcontroller was chosen for the simulation system since it is the only device capable of demodulating all 13 channels by itself.

The MC68332 is a highly capable device, featuring a 32-bit central processing unit (CPU) based on the MC68020, a time processor unit (TPU), a queued serial module (QSM), 2KB of standby RAM, built in chip select capability, and a clock. See Figure 8. The MC68332 is fully compatible with the MC68010 and most of the MC68020 instruction sets. The TPU and QSM are stand-alone subsystems inside the MC68332. The TPU, with its own execution unit, can operate independently of the CPU. It also has its own data storage RAM and microcode ROM. The microcode has a period/pulse width accumulator algorithm which provides the desired pulse-width demodulation function for the simulation system. The TPU controls 16 independent channels, each with a dedicated I/O pin.

At the time this project began, the MC68332 was only available in sample quantities as a preproduction part. Further, it was only available as a surface mount device, making it very difficult to use on a wire-wrap board. To assist developers, however, Motorola offers an M68332 Business Card Computer (BCC). This is a 2.25 inch by 3.5 inch printed-circuit board with an MC68332, 64k x 16-bit EPROM, 32k x 16-bit RAM, RS-232C I/O port, and interface connectors for connecting the BCC to a target system. See Figure 9. The EPROM contains the M68332BUG Debug Monitor, which is a debug tool used to develop systems built around the MC68332. The BCC is used on the pulse-width demodulation board for the simulation system.

The BCC is very useful for developing software for the MC68332 and

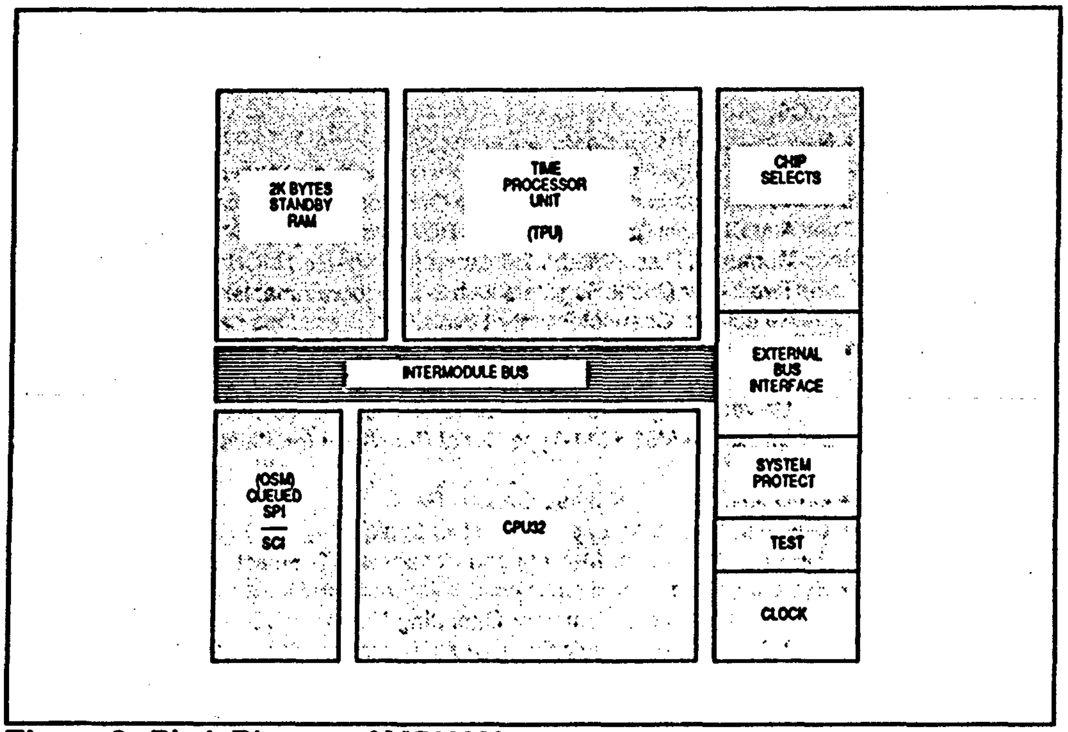


Figure 8: Block Diagram of MC68332

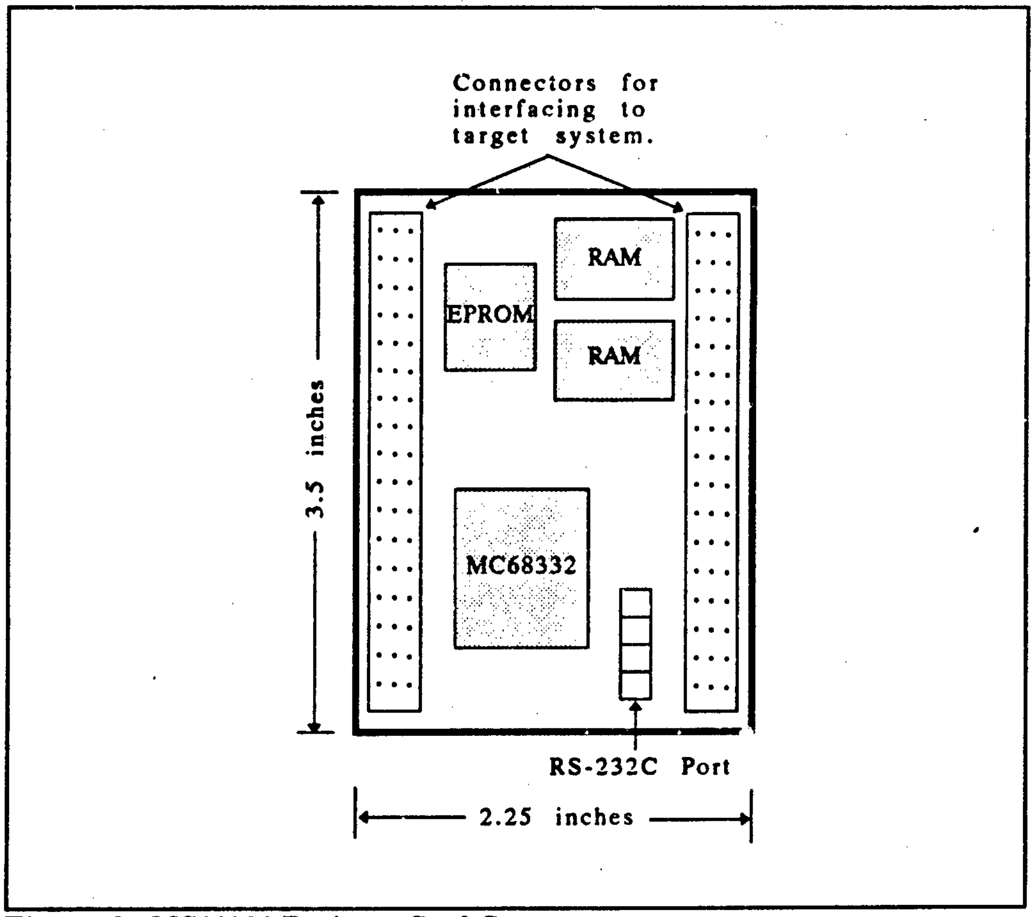


Figure 9: MC68332 Business Card Computer

integrating it into a target system (such as the pulse-width demodulation board for this simulation system). However, it also puts constraints on the developer. The debug monitor is useful for debugging programs, but forces developers to work within the confines of its memory map. It also forces the use of the MC68332's chip select feature. The chip select feature sets 11 of the MC68332's pins to be chip select pins rather than their normal function (address lines A23-A19, function codes FC2-FC0, and bus control lines Bus Grant Acknowledge (BGACK* - the asterisk indicates an active low signal), Bus Grant (BG*), and Bus Request (BR*)). This prevents the use of straight address decoding for off-BCC peripherals and makes it difficult to use peripherals which normally request the local bus for access to board RAM and ROM (separate from BCC RAM and ROM, which are only accessible to the BCC's MC68332).

SECTION 4 - PULSE-WIDTH DEMODULATION BOARD DESIGN

4.1 PULSE-WIDTH DEMODULATION BOARD REQUIREMENTS

The purpose of the pulse-width demodulation (PWD) board is to convert the 13 channels of pulse-width modulated control surface commands generated by the flight computer into digital form. The converted commands must then be made available to the VME processor board for use by the vehicle simulation software.

Since a VMEbus architecture was chosen for the simulation system, the PWD board must be a VME size board (9.2 inches by 6.3 inches) and interface to the VMEbus. The PWD board must be capable of storing program code for the MC68332, which dictates the use of an EPROM. Though the BCC has its own EPROM, it is surface mounted to the BCC and contains the debug monitor code. Trying to reprogram the BCC EPROM would sacrifice its main feature, the debug monitor. Therefore, a separate EPROM must be part of the PWD board.

The PWD board must make the converted pulse-width data available to the 68030 processor board. This requires some type of RAM for storing the data and a VMEbus interface. While the BCC has some on-BCC RAM, it is not accessible to any device but the MC68332 due to the design of the BCC. However, just putting a RAM on the PWD board does not solve the problem. The RAM must be accessible by both the BCC for storing data and the 68030 processor board for reading data. An ordinary RAM, with a single set of address and data lines, depends on the bus control lines to prevent data access collisions (two devices trying to read memory at the same time, producing erroneous results). The BCC cannot use its bus control lines (Bus Request (BR*), Bus Grant (BG*), and Bus Grant Acknowledge (BGACK*)) since they are set by the debug monitor to be chip select lines and hardwired to the on-BCC RAM. This problem was solved by using a dual-port RAM (DPRAM). A DPRAM solves the local bus collision problem since it has two sets of address and data lines, permitting two devices to access its memory simultaneously without using bus control lines. It also permits zero wait access when each device tries to access different data locations at the same time.

4.2 TOP LEVEL DESIGN

A functional block diagram of the PWD board is shown in Figure 10. Since the time processor unit (TPU) converts signals into word-wide data (16-bits) and the BCC defaults to word addressing, a 16-bit wide EPROM and DPRAM were chosen to simplify their interface to the BCC and the VMEbus.

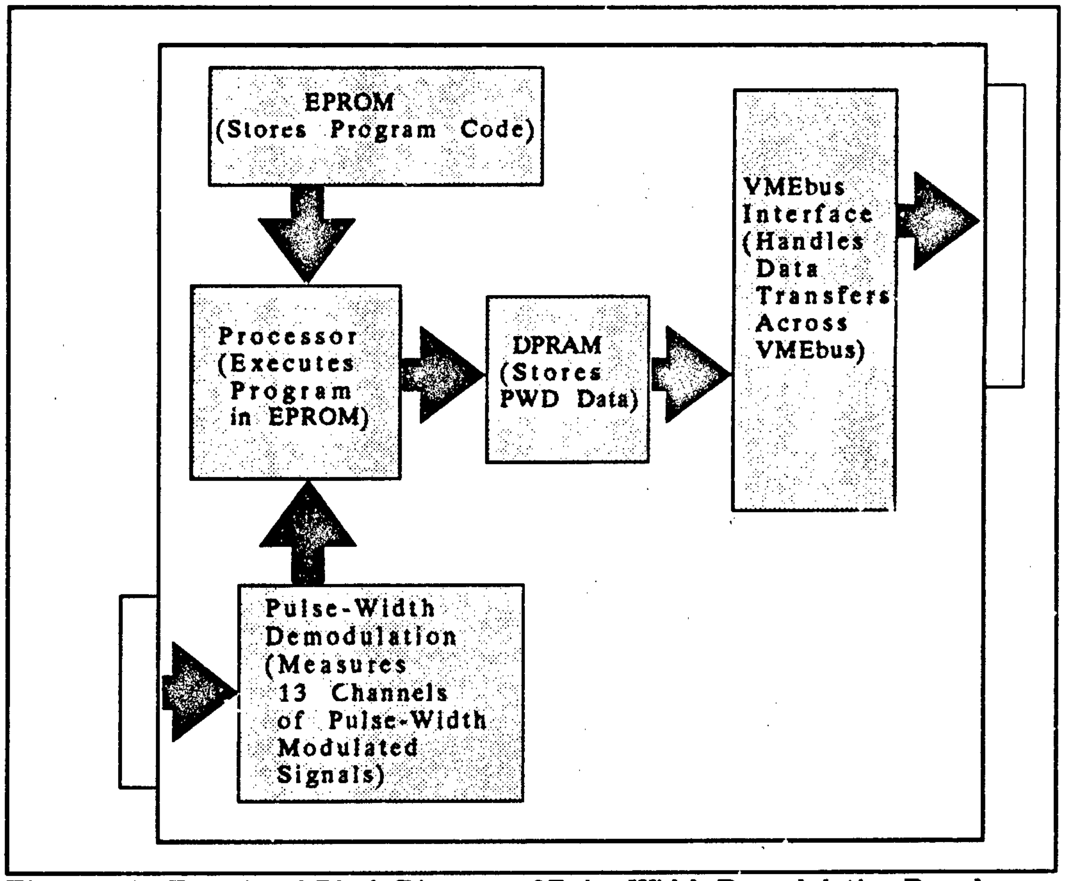

EPROM (Stores Program Code)
Processor (Executes Program in EPROM)
DPRAM (Stores PWD Data)
VMEbus Interface (Handles Data Transfers Across VMEbus)
Pulse-Width Demodulation (Measures 13 Channels of Pulse-Width Modulated Signals)

Figure 10: Functional Block Diagram of Pulse-Width Demodulation Board

The VMEbus Specification (ANSI/IEEE STD 1014) has many interfacing requirements for its 101 signal lines. There are strict requirements for driving and loading each of the five types of signal lines: totem-pole high current, totem-pole standard, three-state high current, three-state standard, and open-collector. There are also strict timing requirements on responses to bus control lines and interrupt lines, as well as daisy-chaining requirements for interrupt signals. Fortunately, there is a device (only one) on the market which greatly simplifies interfacing to the VMEbus: the Motorola MVME6000. With the exception of ACFAIL*, A31-A08, and D31-D08, the MVME6000 directly connects to all VMEbus signal lines. This saves considerable board space and eliminates the need to design a VMEbus interface with discrete components which meets all the VMEbus Specification requirements for timing, sourcing, and loading of signal lines. The PWD board uses the slave mode of the MVME6000, since the PWD board never initiates a VMEbus transfer. If future enhancements are needed, the MVME6000 provides full bus mastership and bus controller capability. Though some of the capabilities of the MVME6000 are not currently implemented, the savings in board space alone justify its use.

4.3 OPERATION OF PULSE-WIDTH DEMODULATION BOARD

The MC68332 executes a program stored in the PWD board's EPROM. That program initializes the PWD board and sets the MC68332 to take in all the pulse-width signals, convert them to digital data, and place the data in a table in the DPRAM. This continuous process takes place on the left half of the PWD board.

On the right half of the PWD board, the MVME6000 responds to periodic requests (every ten milliseconds) from the 68030 processor board to read the pulse-width data in the DPRAM. The MVME6000 translates the VMEbus read request into a local bus read to access the DPRAM. Once initialized, the two sides of the PWD board operate independently. This allows for very fast data transfers on both sides since there is never any local bus arbitration required when accessing the DPRAM (this is what DPRAMs are designed for - transparent access to the same data by separate devices).

4.4 DECODING THE MC68332 ADDRESS LINES

In order to use the BCC and its debug monitor, all peripheral devices on the PWD board must be mapped into the address space defined by the BCC's debug monitor. As shown in the PWD board's address map, Figure 11, the EPROM, DPRAM, and MVME6000 are integrated into the available BCC address space.

The MC68332 has a feature which converts A23-A19, FC2-FC0, BR*, BG*, and BGACK* signal lines to dedicated chip select lines. A chip select option register for each chip select line can be set to define a base address and a block size for which the associated chip select pin will be asserted. The debug monitor and the BCC use this feature. The debug monitor sets up the chip select option lines to enable on-BCC peripherals (RAM and EPROM). In order to use the BCC, the PWD board must be designed so that addressing its peripheral devices is compatible with the BCC's chip select definitions. The PWD accomplishes this by using a programmable logic device, the Lattice Corporation G20V8A Generic Array Logic (GAL), to decode a combination of address lines and BCC defined chip select lines (the address decoder GAL is U1 on the schematics in Appendix A). The GAL then generates the appropriate peripheral chip select signals. The GAL equations for address decoder GAL U1 are in Appendix F.

4.5 INTERFACING THE MC68332 TO THE MVME6000

The MVME6000 requires an on-board processor for its initialization. It cannot be initialized by an off-board device since it does not handle the VMEbus control lines until it is initialized. Thus the MC68332's address, data, and control lines (R/W*, DS*, AS*, SIZ0, SIZ1, CLKOUT, DSACK0*, DSACK1*, HALT*, and RESET*) must be connected to the MVME6000. This presents a problem, since the MC68332's (as well as the MVME6000's) address and data lines must also be connected to the DPRAM. A direct connection between the MC68332 and the MVME6000 would

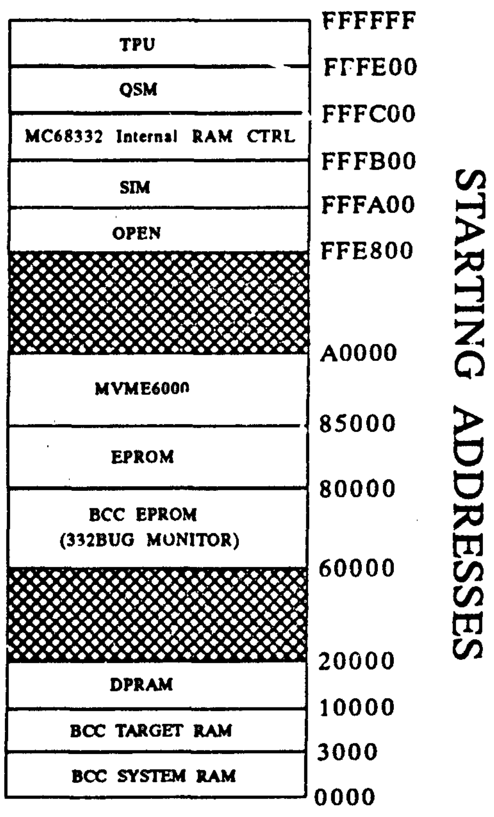


Figure 11: MC68332 Memory Map for PWD Board

prevent the desired transparent, conflict free access to the DPRAM. The design solution to this probl ɜm, as shown in Figure 12, was to isolate the MC68332 from the MVME6000 through bi-directional buffers (74245's). These buffers are address decoded on the PWD board to the same address range as the MVME6000, so they are only enabled when the MC68332 reads from and writes to the MVME6000. Since the MVME6000 has no need to initiate access to the MC68332, the buffers are wired to only allow MC68332 to MVME6000 accesses. This is not a limitation, since without the local bus control lines, the MVME6000 could not initiate access to the MC68332 anyway. If the PWD board was redesigned to use the MC68332 by itself (without the BCC), it would not be difficult to implement two way access if it was desired.

The 74245 buffers do a good job of establishing a connection when the MC68332 addresses the MVME6000 and disconnecting the signals when the MC68332 is not addressing the MVME6000. However, they impose one additional complication. The DSACKx* signal (DSACKx* refers to the DSACK0* and DSACK1* lines, which

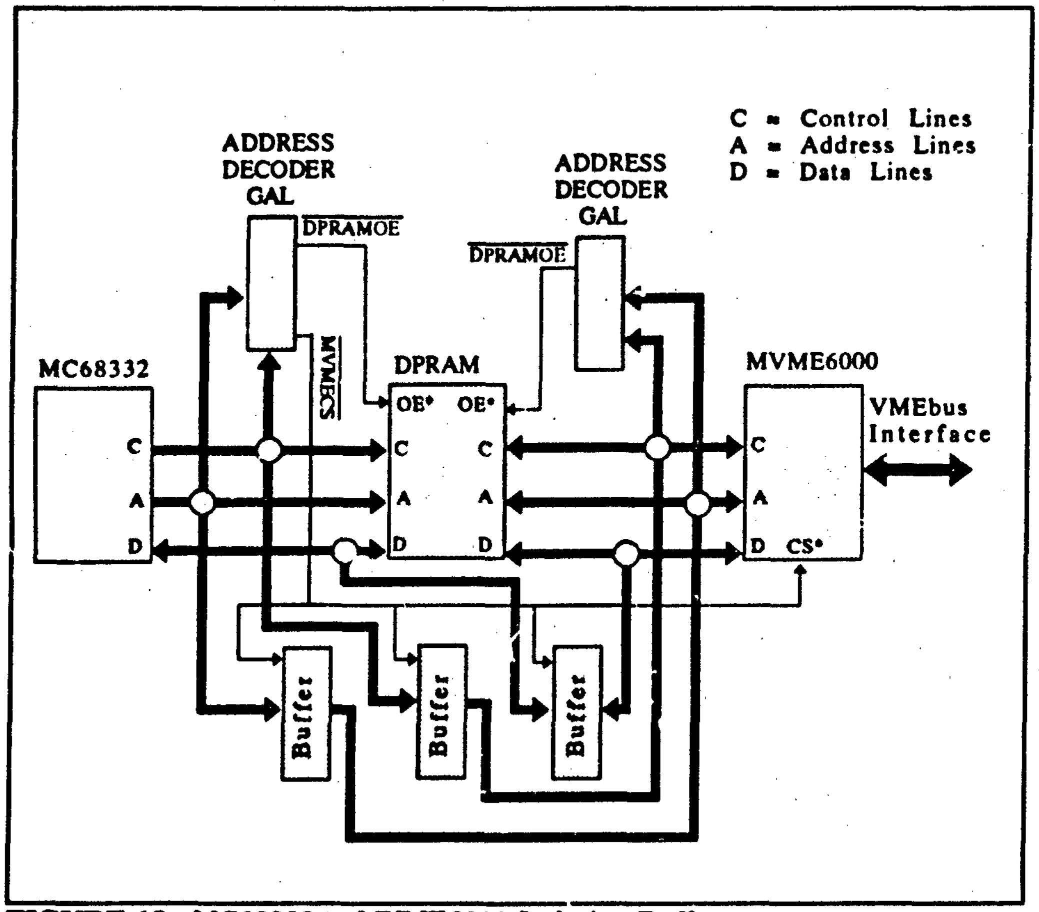


FIGURE 12: MC68332 to MVME6000 Isolation Buffers

in their asserted state will always be DSACK0* = 1 and DSACK1* = 0 (word size acknowledge) for this project) to the MC68332 requires an open-collector driver, but the DSACKx* signal out of the 74245 cannot be fed directly into an open-collector output buffer since when the 74245's are not enabled, their pins are in a high-impedance state. The high-impedance state is interpreted by an open-collector output buffer as a low input, which sets the buffer output to low, continuously asserting the DSACKx* line. This would prevent the MC68332 from reliably writing to and reading from any peripheral device, since the MC68332, seeing DSACKx* asserted, would prematurely end the bus cycle. To solve this problem, another signal, which is only valid when the MVME6000 is accessed but does not pass through the buffers, is needed to determine when DSACKx* from the MVME6000 is valid. This requirement is satisfied by the MVME6000 chip select signal (MVMECS*). The logic required for valid DSACKx* generation is shown in Figure 13.

By inverting MVMECS* and the MVME6000's DSACKx* signal, and then feeding them into an open-collector output NAND gate, a reliable DSACKx* signal is generated to the MC68332.

		(after inverter)		
MVMECS*	DSACKx*	MVMECS	DSACKx	Desired Output
1	1	0	0	1
1	0	0	1	1
0	1	1	0	1
0	0	1	1	0

NAND
Logic

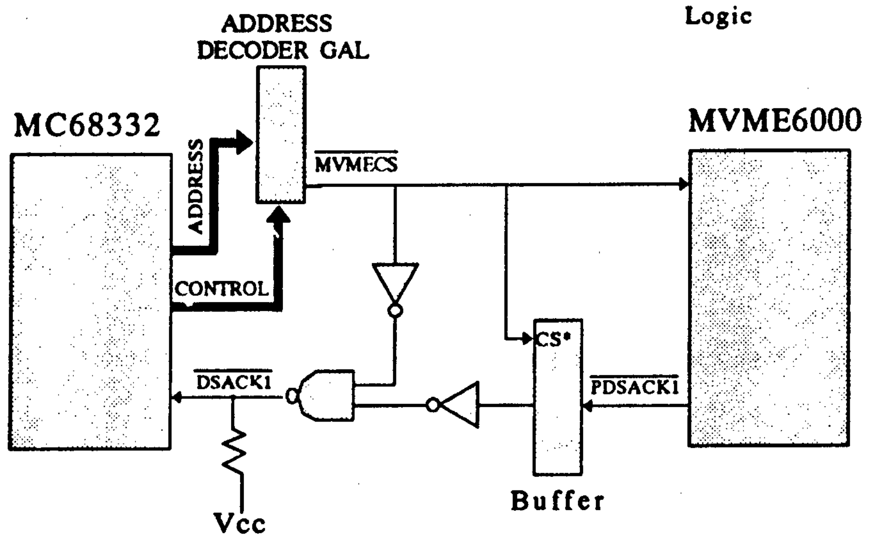


FIGURE 13: MVME6000 DSACKx* Generation Logic

4.6 INTERFACING THE MC68332 TO THE DPRAM

The DPRAM is an Integrated Device Technology IDT7133. It has 55 nanosecond (ns) access time and can automatically arbitrate when both sides request data simultaneously. If simultaneous requests are to separate addresses, both will be serviced within the device's normal 55 ns response time. If simultaneous requests are to the same address, the DPRAM will arbitrate between the two requests, granting one and asserting a BUSY* signal to the other. Since the DPRAM does not assert any type of data transfer acknowledge signal, DSACKx* must be generated for the DPRAM (both right side and left side) by the PWD board since both the MC68332 and the MVME6000 require DSACKx*.

The maximum response times for the DPRAM are broken down as follows:

Read Cycle (BUSY* negated)	Read Cycle (BUSY* asserted)	Write Cycle
55 ns	80 ns	40 ns

The DPRAM has five control lines which are duplicated for each side: R/W* Upper Byte (UB), R/W* Lower Byte (LB), Chip Enable* (DPRAMCE*), Output Enable* (DPRAMOE*), and Busy*. Since all reads and writes to the DPRAM are words (16-bits), the R/W* UB and R/W* LB are tied together, forcing the DPRAM to always perform word-wide operations. During either a read or a write, the DPRAMCE* line must be asserted, but the output enable line, DPRAMOE*, only needs to be asserted during a read operation. The DPRAM is enabled by a GAL (U1 on the schematic, Appendix A) which decodes the address and control lines from the MC68332. When a valid address for the DPRAM is decoded, the GAL asserts DPRAMCE* for reads and writes, and DPRAMOE* for reads.

The MC68332 is operating at 16.77 MHz, producing a clock period of 59.6 ns. As shown in the timing diagram in Figure 14, it takes one clock period after DSACKx* is asserted for the MC68332 to latch in data during a read cycle. Since the DPRAM's response time of 55 ns is faster than the MC68332 latch time of 59.6 ns, it appears that DSACKx* can be generated immediately from the DPRAM's left chip enable signal DPRAMLCE*. However, when a memory access conflict occurs, it takes the DPRAM

a maximum of 35 ns to assert BUSY*. DSACKx* must not be asserted until valid data is guaranteed to be on the bus when the MC68332 latches it in. Therefore, DSACKx* generation must be delayed 35 ns so that BUSY* can be asserted if there is a conflict. When a conflict occurs, DSACKx* must be further delayed until BUSY* is negated. Once BUSY* is negated, DSACKx* may be asserted immediately since the DPRAM will have valid data on the bus before the MC68332 can latch it in. DSACKx* generation is accomplished by feeding the DPRAMLCE signal into at least a 35 ns delay, and then NANDing it with BUSY*. This logic is shown in Figure 15. The exact timing delay of DPRAM DSACKx* will be addressed in the MC68332 Interface to EPROM section.

4.7 INTERFACING THE MVME6000 TO THE DPRAM

The MVME6000 latches data from the local bus faster than the MC68332, though its user manual does not specify just how fast it is. DSACK1* to the MVME6000 for DPRAM accesses must be delayed the full 55 ns to ensure that the MVME6000 latches in valid data. This delay already encompasses the time it takes the DPRAM to assert BUSY*, so this is not a concern for this side. As on the left side, the right side DSACKx* is generated by delaying the DPRAMRCE signal. This signal is delayed by looping multiple times through the address decoding device (a programmable generic array logic device), and then further delayed by two AND gates, before it is NANDed with BUSY* to produce DSACKx* to the MVME6000. The combination of these device delays (the time it takes a signal to pass through a device) are enough for the DPRAM to provide valid data to the MVME6000 and VMEbus. Note that the NAND gates which generate DSACKx* are open-collector output (7403 devices), and the DSACKx* lines have the required pull-up resistors. Open-collector signal lines permit multiple devices to drive the line since each device is effectively disconnected from the line when not asserting it (pulling it low).

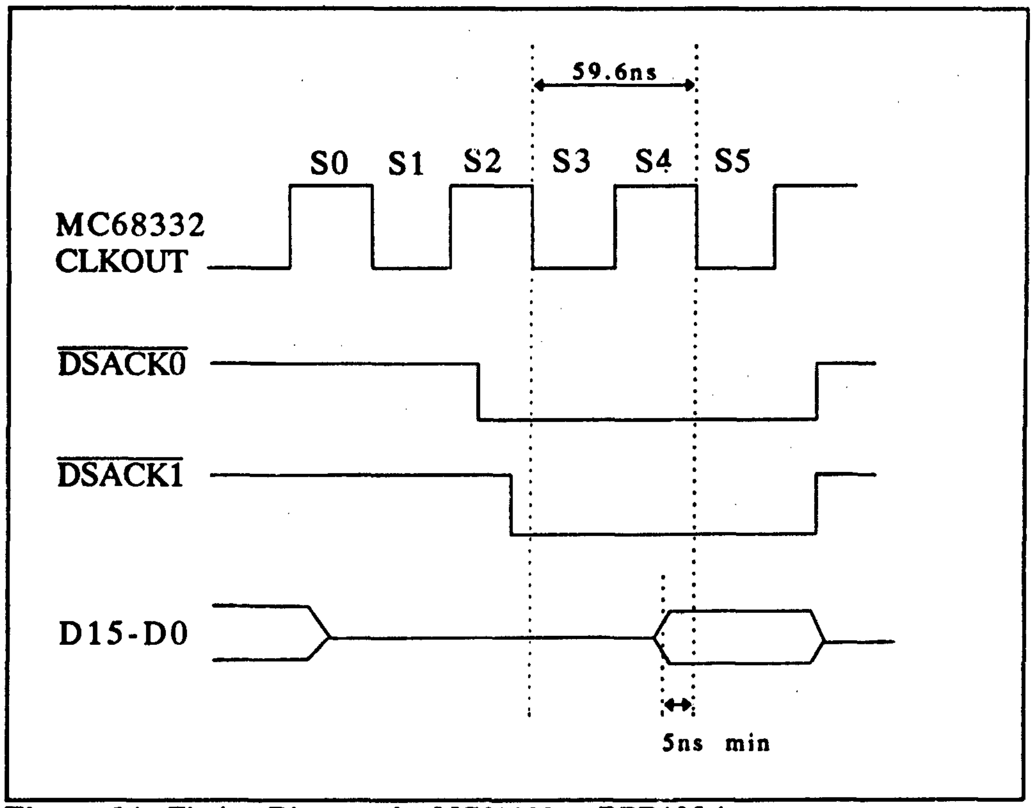


Figure 14: Timing Diagram for MC68332 to DPRAM Access

4.8 INTERFACING THE MC68332 TO THE EPROM

The Advanced Micro Devices Am27C1024 EPROM was chosen to store the program code for the MC68332. The Am27C1024 (referred to simply as EPROM) is the first EPROM device to offer the desired 16-bit inputs and outputs. This simplifies the connection to the MC68332 since it is set to read 16-bit words. Access time is 150 ns (fast by EPROM standards). To access the EPROM, a GAL (U1 on the schematic, Appendix A) is programmed to decode addresses put out by the MC68332 and generate the EPROM output enable (EPROMOE*) signal when an address mapped to the EPROM address range appears on the address bus.

Like the DPRAM, the EPROM does not generate any type of "data ready" signal, requiring other logic to generate DSACKx*. The EPROM's 150 ns access time is much slower than a normal MC68332 read cycle, requiring a delay for DSACKx* generation. The minimum DSACKx* delay is the EPROM's 150 ns maximum access time minus the 59.6 ns latch time of the MC68332, which equals about 91 ns. A GAL is used on the PWD board (U3 on the schematic in Appendix A) to generate the appropriate DSACKx* delay. With a clock signal input on pin 1, the GAL can be programmed to clock its outputs. Normally, the GAL's outputs would reflect the state

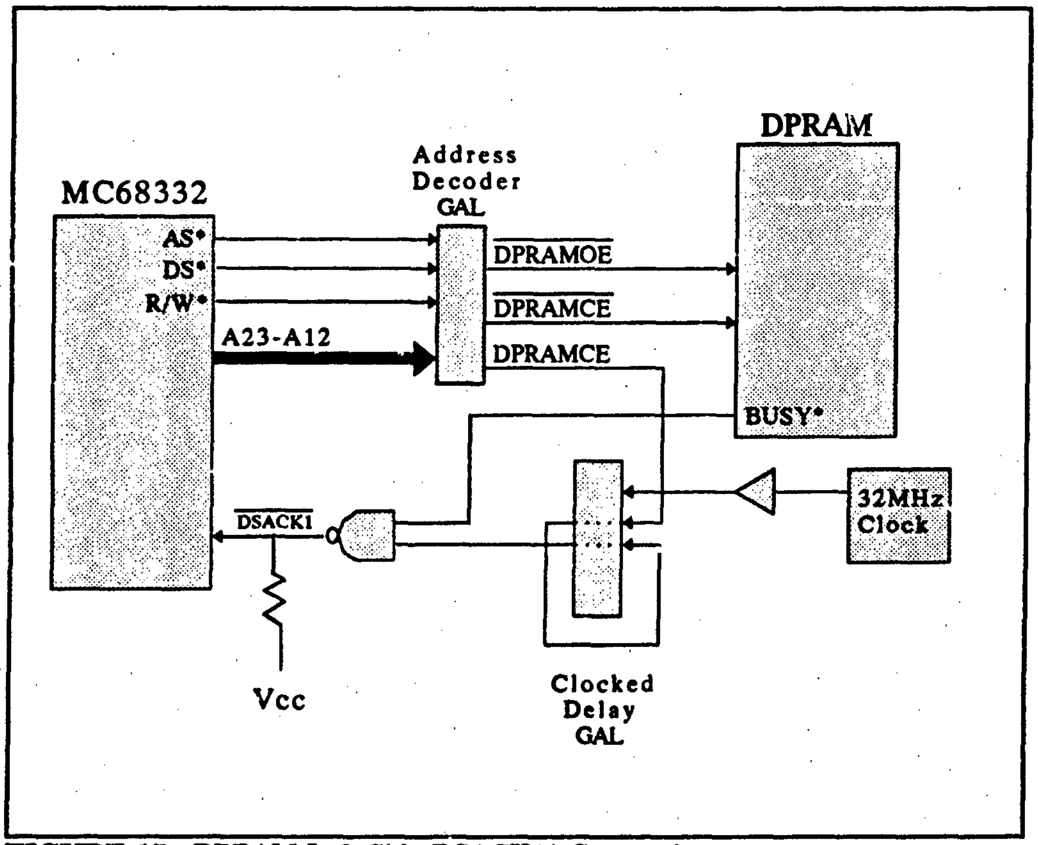


FIGURE 15: DPRAM Left Side DSACK1* Generation

of its inputs after only 10 ns (the GAL has a 10 ns signal propagation delay from input to output). Using clocked outputs, the outputs will not reflect the state of the inputs until the falling edge of the next clock. The minimum and maximum clocked propagation delays are depicted in Figure 16. The logic for generating DSACKx* for the EPROM is very similar to that of the DPRAM. The EPROM output enable line is also used to generate DSACKx* for the EPROM. In order to maximize the use of each GAL, and to minimize board space used, the same clocked GAL is used to generate the appropriate DSACKx* delays for both the EPROM and the DPRAM.

To determine the clock value for the GAL, both the DPRAM and the EPROM DSACKx* delay requirements should be considered. The DSACKx* delay for both devices should be as close as possible to their respective minimum required delays. It is not a problem for DSACKx* to be delayed longer than necessary, but it does slow down the MC68332's program execution each time that device is accessed. Recall that the DPRAM requires a minimum delay of 35 ns, and the EPROM requires a minimum delay of 100 ns. These two delays are close enough that it shouldn't be a problem for the same GAL to generate both delays.

Efficient access to the DPRAM is the primary consideration for choosing a clock frequency for two reasons: (1) data will be stored in the DPRAM on a regular basis,

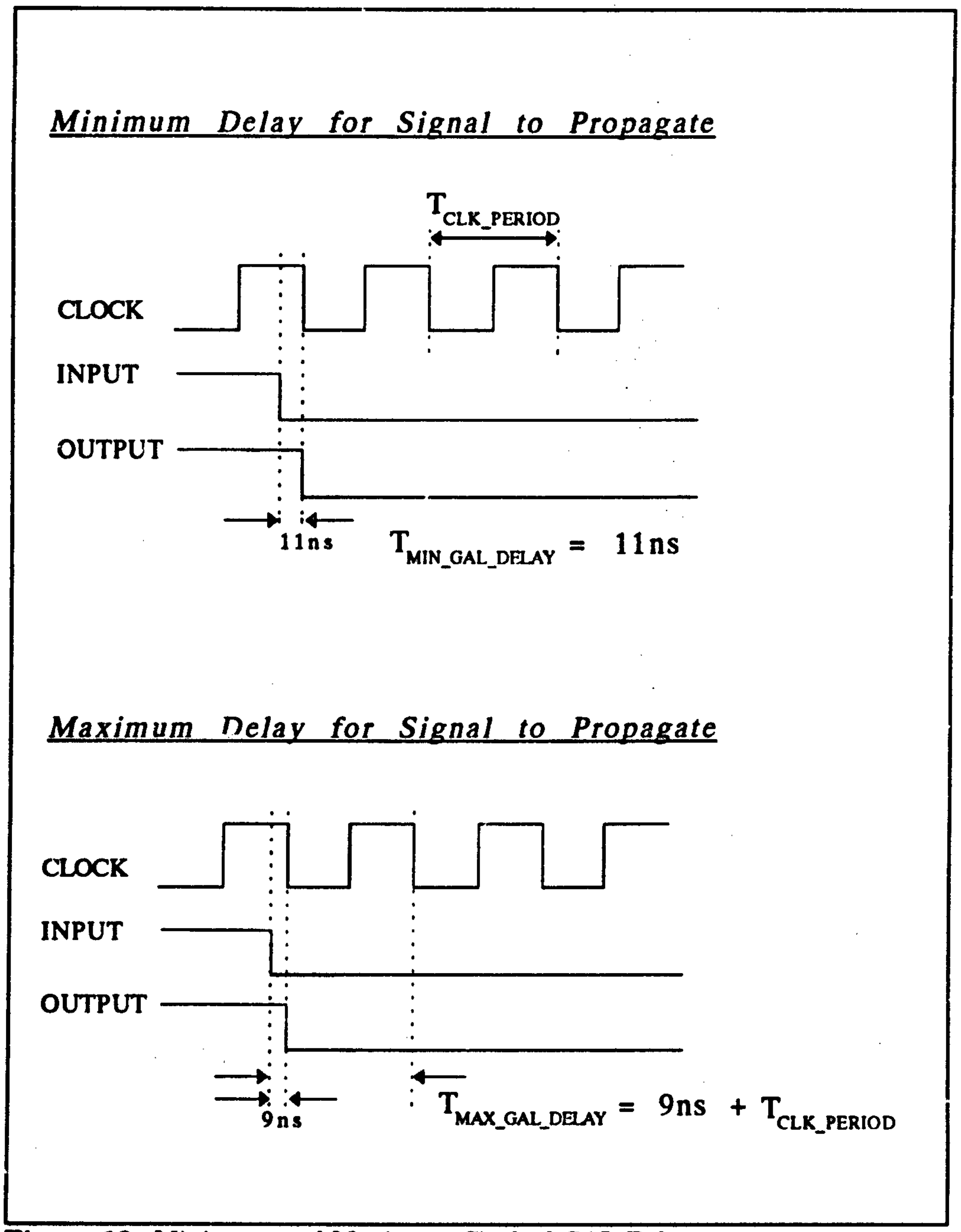

Minimum Delay for Signal to Propagate
TCLK_PERIOD
CLOCK
INPUT
OUTPUT
11ns
TMIN_GAL_DELAY = 11ns
Maximum Delay for Signal to Propagate
CLOCK
INPUT
OUTPUT
9ns
TMAX_GAL_DELAY = 9ns + TCLK_PERIOD

Figure 16: Minimum and Maximum Clocked GAL Delays

and (2) the EPROM program code can be copied from EPROM to RAM for faster execution (after which the EPROM would no longer need to be accessed). In calculating the minimum DSACKx* delay time for the DPRAM, there is the minimum GAL propagation delay, T_{GAL_MIN}, of 11 ns (see Figure 16) plus the number of delay loops that the signal is fed back through the GAL (n) multiplied by the GAL clock period:

DPRAM DSACKx* Minimum Delay Equation:

$$T_{GAL\_MIN} + (n * T_{GAL\_CLK\_PERIOD}) >= 35 \text{ ns}$$

The closest commonly available clock frequency to solve this equation is 32 MHz. A 32 MHz clock has a period of 31.25 ns ($1/(32x10^6)$ = 31.25 ns). Solving the DPRAM DSACKx* delay equation:

$$11 \text{ ns} + (n * 31.25 \text{ ns}) >= 35 \text{ ns}$$
$$n * 31.25 \text{ ns} >= 24 \text{ ns}$$
$$n >= .77$$

Since n must be a whole number, n = 1 delay loop through the GAL. The resulting minimum DPRAM DSACKx* delay is:

$$11 \text{ ns} + 31.25 \text{ ns} = 42.25 \text{ ns}$$

This is close to the minimum delay of 35 ns, and allows a safety margin of about 7 ns. The maximum DPRAM DSACKx* delay is:

DPRAM DSACKx* Maximum Delay Equation:

$$T_{GAL\_MAX} + (n * T_{GAL\_CLK\_PERIOD}) = \text{Maximum DSACKx* Delay}$$

From Figure 16, T_{GAL_MAX} = 9 ns + $T_{GAL\_CLK\_PERIOD}$, so the equation is:

$$(9 \text{ ns} + 31.25 \text{ ns}) + (1 * 31.25 \text{ ns}) = \text{Maximum DPRAM DSACKx* Delay}$$
$$= 71.5 \text{ ns}$$

At about double the minimum required DPRAM DSACKx* delay, 71.5 ns appears to be a long time. However, it only incurs one additional MC68332 clock period delay (wait state), and since the minimum delay is close to the lower bound of 35 ns, a slightly faster clock frequency would not improve the delay time by a significant amount.

The 32 MHz clock turns out to be a very good selection for the EPROM DSACKx* delay, too. The EPROM DSACKx* delay must also take into account the 30 ns high-to-low transition delay of the open-collector buffer shown in Figure 17. Note that the NAND gate on the DPRAM DSACKx* line was not considered since the objective of delaying DPRAM DSACKx* is to wait for DPRAM BUSY* to be valid, which occurs prior to the signal passing through the NAND gate. The minimum EPROM DSACKx* delay is:

$$T_{GAL\_MIN} + T_{BUFFER\_DELAY} + (n * T_{GAL\_CLK\_PERIOD}) >= 91 \text{ ns}$$
$$11 \text{ ns} + 30 \text{ ns} + (n * 31.25 \text{ ns}) >= 100 \text{ ns}$$
$$n >= 1.9$$
$$n = 2$$

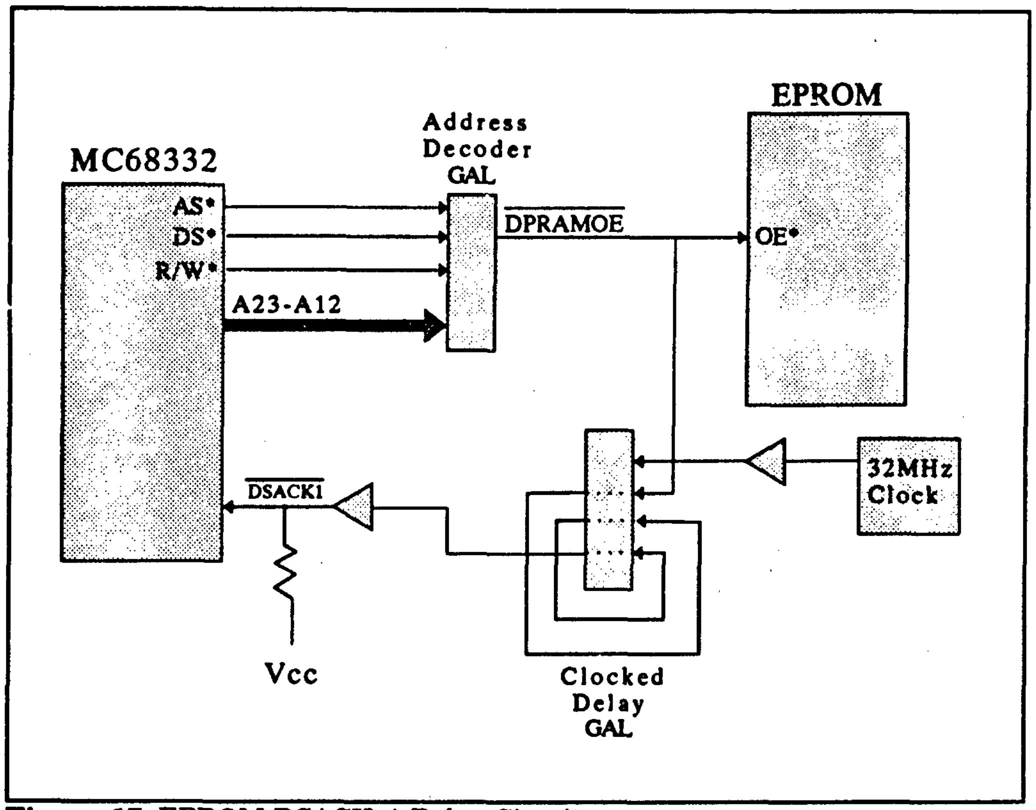


Figure 17: EPROM DSACKx* Delay Circuit

The EPROM DSACKx* delay signal must feed back through the GAL twice. This yields the following minimum and maximum delay times:

Minimum EPROM DSACKx* Delay:

$$T_{GAL\_MIN} + T_{BUFFER\_DELAY} + (n * T_{GAL\_CLK\_PERIOD}) = \text{delay}$$
$$11 \text{ ns} + 30 \text{ ns} + 62.5 \text{ ns} = 103.5 \text{ ns}$$

Maximum EPROM DSACKx* Delay:

$$T_{GAL\_MAX} + T_{BUFFER\_DELAY} + (n * T_{GAL\_CLK\_PERIOD}) = \text{delay}$$
$$40.25 \text{ ns} + 30 \text{ ns} + 62.5 \text{ ns} = 133 \text{ ns}$$

These EPROM DSACKx* delay times compare very favorably to the lower bound of 91 ns.

These delays in generating DSACKx* for the EPROM and DPRAM are necessary to insure that valid data is on the bus when the MC68332 latches the data during a read cycle, and so that the MC68332 maintains valid data on the bus until it is written into the DPRAM during a write cycle. The MC68332 has a requirement, though, that DSACKx* be negated within 80 ns of the negation of AS* and DS* to prevent DSACKx* from interfering with the next bus cycle. AS* and DS* are negated when the MC68332 detects the assertion of DSACKx*.

Unless special design considerations are made, the negation of DSACKx* for each peripheral on the PWD board follows the same path as DSACKx* assertion. In calculating DSACKx* negation times for each peripheral, maximum delay times for each contributing device must be used to ensure that DSACKx* will never exceed 80 ns. The first DSACKx* negation delay for the DPRAM and the EPROM is the 10 ns propagation delay of the address decoder GAL (U1 in the schematic, Appendix A; all non-clocked GALs on the PWD board have a propagation delay of 10 ns). DSACKx* generation (and therefore negation) for the DPRAM and the EPROM rely on their chip select lines. Since DS* is part of the address decoder GAL equations for asserting each of the chip select signals, the negation of DS* (along with the negation of the other bus signals by the MC68332) causes the chip select signals to be negated 10 ns later. Following the path for DSACKx* assertion, the next delay is due to the clocked delay GAL (U3 in the schematic) for both the DPRAM and the EPROM. As shown above, the DPRAMCE signal is delayed up to 71.5 ns by the delay GAL, and then another 22 ns by the NAND gate. This totals 103 ns maximum DSACKx* negation delay for the DPRAM, which is 23 ns too long:

10 ns address decoder GAL delay
+ 72 ns maximum clocked GAL delay
<u>+ 22 ns</u> NAND gate
= 103 ns maximum unmodified DSACKx* negation delay

To shorten this negation delay, the U3 delay GAL is programmed so that the negation of the DPRAMCE* input will negate the DPRAMCE output on the next clock cycle. It won't wait for the DPRAMCE* input to be looped once back through the delay GAL, as it does for DPRAMCE assertion. This saves 31 ns, making the maximum DSACKx* negation delay 72 ns, which is within the 80 ns maximum negation requirement.

The EPROM DSACKx* negation delay is handled the same way. In calculating DSACKx* negation delay, though, the transition of the open-collector output buffer is from low-to-high, which is only 10 ns. So the EPROM DSACKx* negation delay is automatically 20 ns faster than its assertion time. The maximum negation delay without modifications would be:

10 ns address decoder GAL delay
<u>+ 113 ns</u> maximum clocked GAL and buffer delay
= 123 ns maximum unmodified DSACKx* negation delay

which is 43 ns too slow. This delay is shortened in the same manner as for the DPRAM above. The clocked delay GAL is programmed so that negation of the EPROMOE* input will negate the EPROM DSACKx* signal on the next falling clock transition, rather than waiting for two feedback loops as in EPROM DSACKx* assertion. This saves 62 ns, making the maximum EPROM DSACKx* negation delay 123 ns - 62 ns = 61 ns. This well within the 80 ns maximum negation delay and is even faster than the DPRAM DSACKx* delay due to the buffer transition delay being faster than the NAND gate.

4.9 INTERFACING THE MVME6000 TO THE VMEBUS

As previously mentioned, all of the VMEbus signals except ACFAIL*, A31-A08, and D31-D08, can be directly connected to the MVME6000. To enable the MVME6000 on the VMEbus side, an address decoder is required. A GAL on the PWD board (U12 on the schematics in Appendix A) is programmed to enable the MVME6000 for VMEbus addresses 400000hex to 410000hex. The VMEbus address map is shown in table 18. Since a VMEbus 24-bit standard address range was selected, the GAL drives the MVME6000's MATCH24* line to enable it. If desired, the MVME6000 could have been mapped into the VMEbus 32-bit extended address range or the VMEbus 16-bit short address range by having the GAL decode the appropriate address lines and drive MATCH32* or MATCH16*. The address decoder GAL also decodes address 410000hex to 41xxxxhex for the MVME6000's Match Global Control and Status Register (MATCHGCSR*). This signal allows other VMEbus boards to read the MVME6000 GCSR to determine the state of the PWD board (whether it is ready, halted, failed, etc).

The VMEbus signal lines that are not connected to the MVME6000, namely the address and data lines (ACFAIL* is not used by the PWD board), require the appropriate buffers for driving the VMEbus lines. The 74543 buffer is commonly used to drive the VMEbus address and data lines. Additional buffers are needed to latch the input and output data from and to the VMEbus. The MVME6000 controls these buffers, as shown in the schematics in Appendix A. Since the PWD board is only using 16-bit data and 24-bit addresses, these are the only VMEbus data and address lines brought onto the board.

Table 2: VMEbus Memory Map

DPRAM	400000 - 40FFFF Hex
GCSR	410000 - 41FFFF Hex
D/A Board	0C0000 - 0C0080 Hex
68030 Board	800000 - FFFFFF Hex

4.10 RESET CIRCUIT AND STATUS LEDS

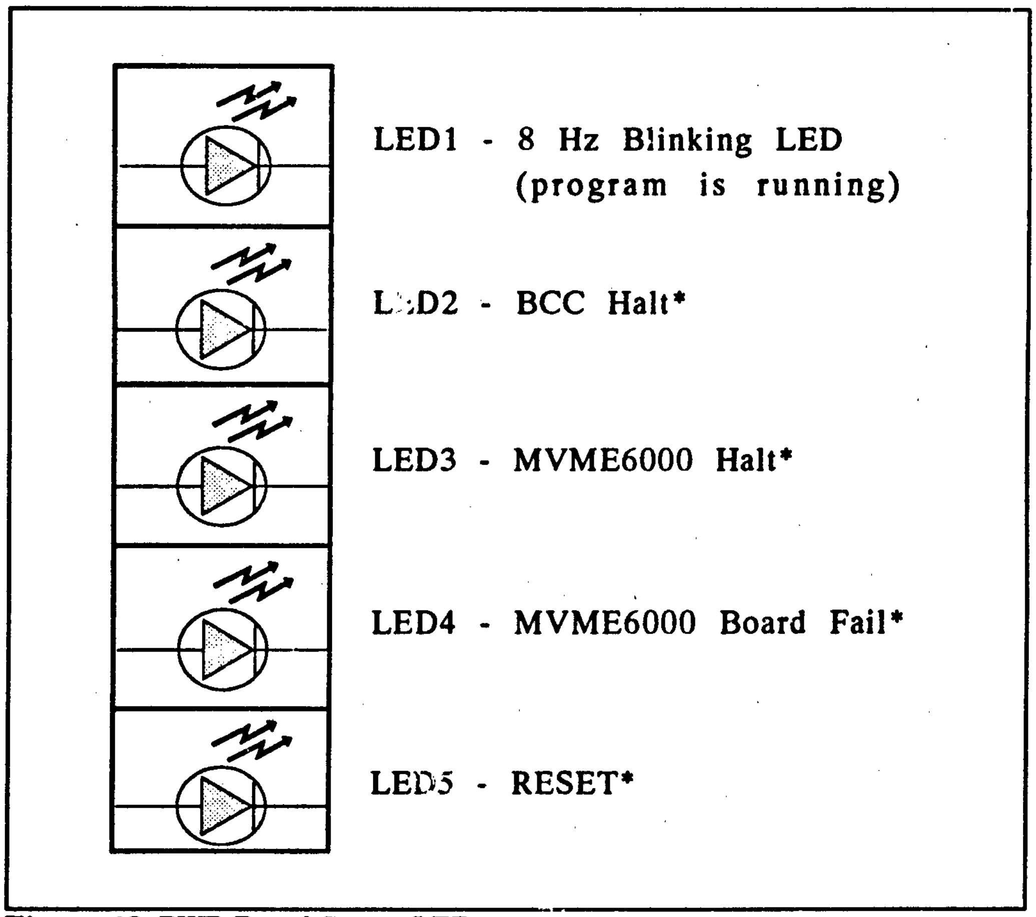
The PWD board has a reset switch that resets the MC68332 and the MVME6000 (the only two devices which have a RESET* line). As shown on sheet 3 of the PWD board schematics in Appendix A, the reset switch is connected to a common 555 timer debounce circuit for reliable operation.

There is a bank a five status LEDs which show the status of the PWD board. Figure 18 shows the order of the LEDs and their meaning. LED1 blinks at an 8 Hz rate when the PWD software is executing. This LED is driven by TPU Channel 15, which is programmed to output a 128 Hz pulse width with 50 percent high time. This is the slowest pulse rate possible from a TPU channel since the TPU is initialized to its fastest clock rate for the most accurate pulse-width measurements. In order to slow down the blinking rate so that it is noticeable (rather than appearing to be a steady "on" condition), the signal is fed into a 74F161 binary counter. The counter divides the frequency by 16, yielding a noticeable blink rate of 8 Hz.

LEDs 2 and 3 are connected to the BCC's and MVME6000's HALT* lines, respectively. If either of these two devices halt, the corresponding status LED will show it.

LED 4 is the MVME6000's board fail signal (BRDFAIL*). When the PWD board is powered-up, LED 4 is initially lit. This reflects the fact that the MVME6000 has not yet been initialized. When the MC68332 initializes the MVME6000, it resets the MVME6000's BRDFAIL* bit in the MVME6000's local control and status register. This turns off LED 4. The state of the BRDFAIL* bit is also reflected in the global control and status register. This allows other VMEbus boards that want to access the PWD board to poll that bit for determining when the MVME6000 has been initialized and is ready for data transfers.

LED 5 is the RESET* LED. Whenever the RESET* line on the PWD board is active, such as when the reset switch is pushed, LED 5 lights up.


LED1 - 8 Hz Blinking LED (program is running)
LED2 - BCC Halt*
LED3 - MVME6000 Halt*
LED4 - MVME6000 Board Fail*
LED5 - RESET*
Figure 18: PWD Board Status LEDs

SECTION 5 - SYSTEM SOFTWARE

5.1 SOFTWARE FOR THE MC68332 PULSE-WIDTH DEMODULATION BOARD

The MC68332 is programmed in Motorola MC68020 assembly language. The program is compiled into Motorola S-3 record format and burned into the AMD 27C1024 EPROM. A functional block diagram of the MC68332 software is shown in Figure 19.

The first thing the program does is set the MC68332 clock register for 16 MHz operation (its fastest option). An area of the DPRAM is then cleared for storing the pulse-width accumulation values. It is these pulse-width values stored in the DPRAM that the 68030 processor board reads across the VMEbus. To help maximize throughput, the interrupt handlers for the TPU pulse-width accumulate channels are copied from EPROM to RAM since code executes faster from RAM than EPROM (due to the slower response time of EPROM). With the interrupt handlers for each pulse-width accumulate channel in place, the program loads the interrupt handler address for each one into the appropriate vector number offset in the interrupt vector table. The program then initializes the TPU. The TPU clock prescalar is set to its fastest mode of operation, providing a maximum resolution of 250 ns for the pulse-width measurements. This is far better than the 2 μs output resolution of the flight computer.

The input/output channels of the TPU are initialized to 13 channels of interrupt driven pulse-width accumulate (demodulate) and one channel of pulse-width modulated output (to drive the blinking status LED). All TPU channels function independently and have their own dedicated input/output pin on the MC68332. Each of the 13 pulse-width accumulate channels are programmed for identical operation, varying only by where they store their pulse-width accumulated data. Each pulse-width accumulate channel starts counting on a low-to-high transition on its input pin, and continues to count until a high-to-low transition is detected. When the high-to-low transition is detected, the channel is programmed to generate an interrupt. The interrupt handler stores the accumulated pulse-width count and clears the channel's interrupt status request so that the channel is ready to interrupt again on the next high-to-low transition. The TPU channels store their count in separate upper byte and lower byte registers. The interrupt handler must copy the lower byte, then check if the count was large enough to flow into the upper byte register. If there is data in the upper byte, it must be copied into the upper byte location where the lower byte was stored, and then the upper byte register must be cleared since the TPU only automatically clears the lower byte. While checking the upper byte for data may seem like unnecessary overhead, it is much faster to check for data and skip storing and clearing the register when no data is present than to default to always storing and clearing the register (unless all pulse-width accumulates overflow into the upper byte).

Lastly, the MVME6000 is then initialized to slave mode. After initializing the

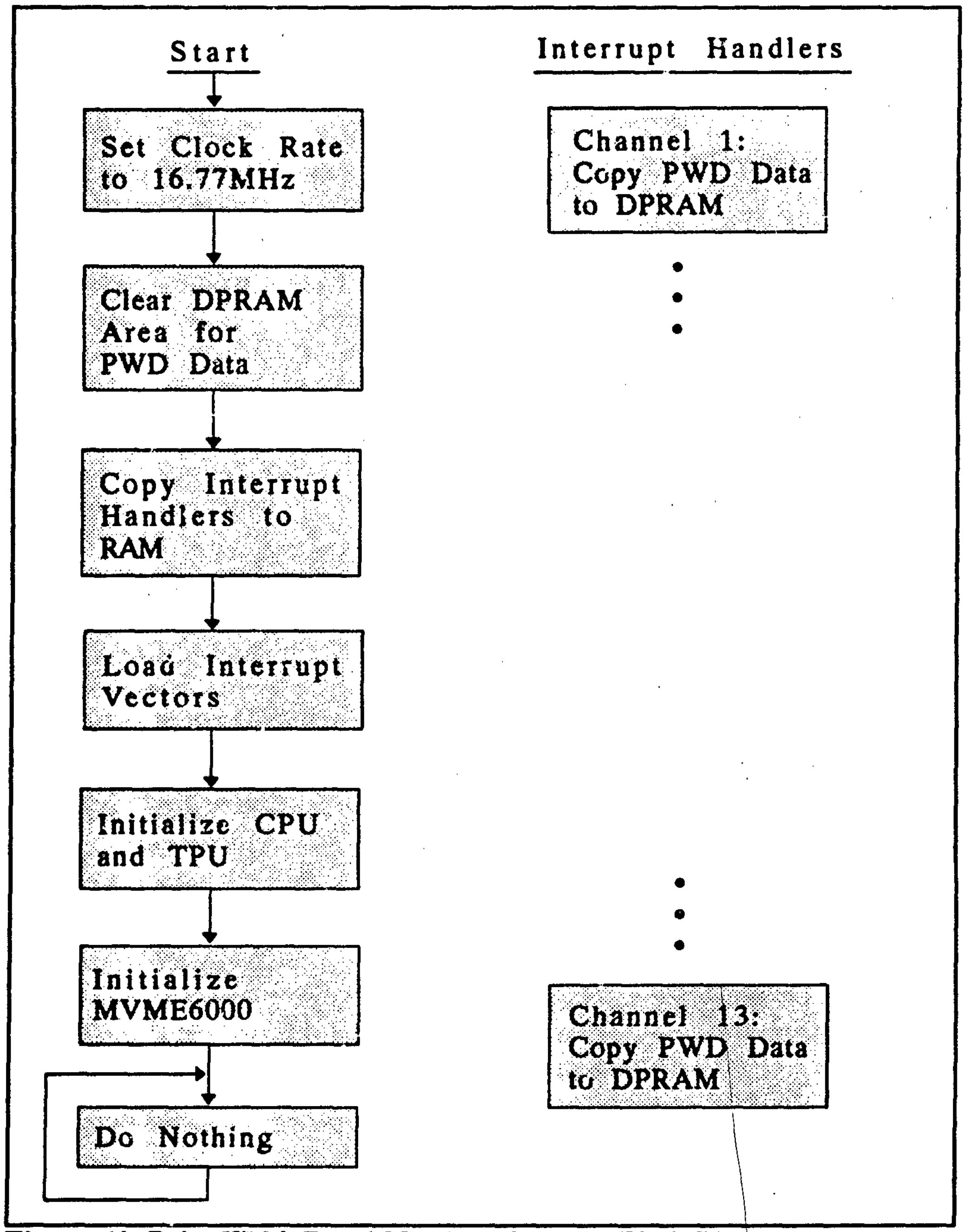

Start
Set Clock Rate to 16.77MHz
Clear DPRAM Area for PWD Data
Copy Interrupt Handlers to RAM
Load Interrupt Vectors
Initialize CPU and TPU
Initialize MVME6000
Do Nothing
Interrupt Handlers
Channel 1: Copy PWD Data to DPRAM
Channel 13: Copy PWD Data to DPRAM

Figure 19: Pulse-Width Board MC68332 Software Block Diagram

TPU and the MVME6000, the program goes into a continuous loop in which it doesn't do anything. The TPU executes independently of the CPU part of the MC68332, so the CPU is left with nothing to do. The CPU is not totally idle, however, since it processes the interrupts generated by each pulse width accumulate channel.

5.2 SOFTWARE FOR THE MC68030 PROCESSOR BOARD

The 68030 is programmed in C. C and assembly were the only compiler and assembler available that could produce files for running on the 68030 board. As with the MC68332, the program is compiled into Motorola S-3 record format and burned into an EPROM so that it can execute on the 68030 board. A functional block diagram is shown in Figure 20.

To guarantee real time response, the vehicle simulation code must execute once every 10 ms. To do this, the program initializes the board's Zilog Z8536 Counter/Timer and Parallel I/O Unit (CIO) to generate an interrupt once every 10 ms. The simulation software then executes in the interrupt handler. As long as the interrupt handler is done executing before the CIO issues the next 10 ms interval interrupt, the simulation software meets its real time execution constraints. Any time that is left over between interrupts is spent displaying control surface positions and sensor output values. This allows an easy visual check of the simulation to see if it is functioning properly.

When the program starts, it first initializes the digital-to-analog (D/A) board for fast output refresh rate (every channel is updated once every 0.85 ms), a 0 to 5-volt output range, and to start scanning and outputting the channels. The program then initializes the CIO to generate an interrupt every 10 ms. The interrupt vector is set to the address of the interrupt handler in a vector definition table which is linked into the program during the link process when generating an executable file.

The simulation software (in the interrupt handler routine "timer3()") first restarts the CIO timer for the next 10 ms interrupt, then reads the demodulated pulse-width data from the DPRAM. This data is converted to "degrees of control surface deflection", and then scaled into the proper range for each control surface. The simulation software then executes a model of the vehicle's actuators. This accounts for the actuator response rate (how long it takes the servo to move from its current position to the commanded position) and delay (how long it takes the servo to start moving). The rest of the simulation code was obtained by converting an existing Fortran software model of the aircraft to C using a translator. The translation was close, but not perfect. Some parts of the translation had to be modified to get the code to execute properly. The translated code perfor ms the following computations:

- Compute the aerodynamic variables based on the state variables (which are computed from the new surface positions and the previous state of the vehicle).

- Compute longitudinal forces and moments.

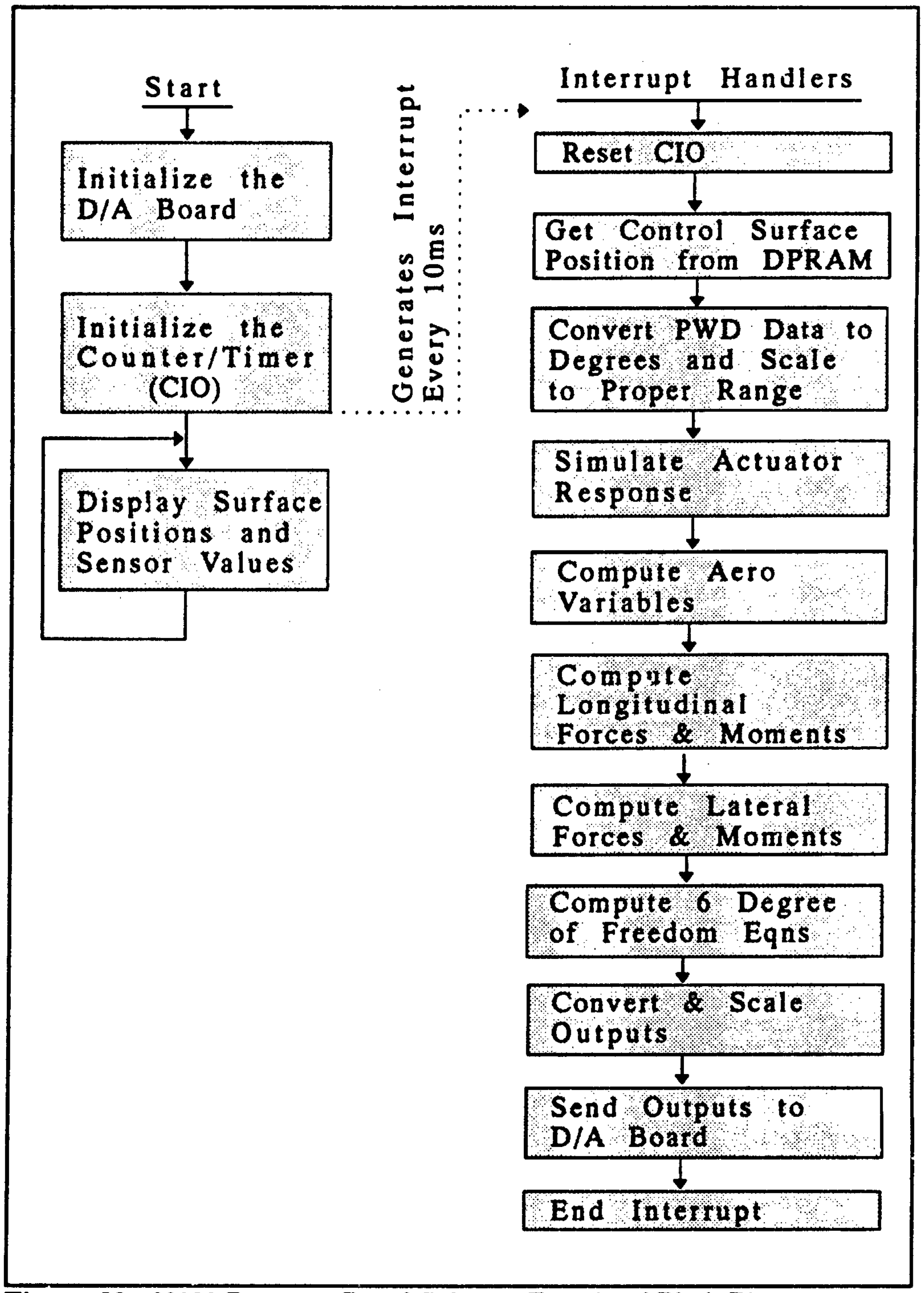

Start
Initialize the D/A Board
Initialize the Counter/Timer (CIO)
Display Surface Positions and Sensor Values
Generates Interrupt Every 10ms
Interrupt Handlers
Reset CIO
Get Control Surface Position from DPRAM
Convert PWD Data to Degrees and Scale to Proper Range
Simulate Actuator Response
Compute Aero Variables
Compute Longitudinal Forces & Moments
Compute Lateral Forces & Moments
Compute 6 Degree of Freedom Eqns
Convert & Scale Outputs
Send Outputs to D/A Board
End Interrupt

Figure 20: 68030 Processor Board Software Functional Block Diagram

- Compute lateral forces and moments.

- Compute the next state of the vehicle based on six degree-of-freedom nonsymmetric rigid body equations of motion.

The interrupt handler converts the computed sensor values (predicted rates, attitudes, air speed, altitude, side-slip) from double precision variables to 12-bit variables. These 12-bit variables are then scaled to meet the output range of the particular sensor they are simulating. This is necessary so that the flight computer gets the same type of signal from the simulation that it would normally get from the vehicle sensors while flying.

SECTION 6 - CONCLUSIONS

6.1 SYSTEM PERFORMANCE

Using a VMEbus Analyzer board, the simulation was timed from the 68030 board's first read from the DPRAM pulse-width data table to the last sensor channel write to the D/A board. This is the time it takes the simulation software to completely execute once. This time was consistently measured at 7.92 ms. To get the entire simulation time, the D/A board output refresh rate must be added:

```
  7.92 ms execution time for simulation software
+ 0.85 ms output refresh rate for D/A board
  8.77 ms simulation execution time
```

The pulse-width conversion process is not figured into this execution time since it is a continuous process. The most recent pulse-width output from the flight computer will be in the DPRAM data table when the simulation software reads it. The 8.77 ms simulation time meets the 10 ms execution rate required for a real time simulation. The remaining 1.2 ms between simulation execution is used by the 68030 to display simulation values to a monitor.

To verify that this system is an accurate simulation of the air vehicle, simulation data was stored in real time to a RAM table. After the pilot commanded a maneuver to the flight computer, the stored simulation data was retrieved and plotted. The control algorithm used in these tests was a rate controller. Three single axis step inputs were commanded. The proper response to these inputs is a sharp change in the pitch, roll, and yaw rates, which should then settle to a rate of zero. The simulation response was plotted (shown in Figure 21) and compared to plots of a verified software model of the aircraft performing the same maneuver. These plots were virtually identical, verifying the accuracy and response rate of the simulation system. These results are as accurate as those provided by hiring specialized facilities to perform simulations. This simulation system provides much faster turn around of simulation data and is more flexible in terms of being able to vary and record any of the simulation parameters.

6.2 LIMITATIONS

The simulation system is limited by the simulation software to certain flight conditions. The simulation software is designed for a limited range of altitudes, rates, airspeeds, and attitudes for the vehicle. The current values for these ranges meet the majority of flying conditions for the unmanned research vehicle. Since this vehicle is flown at low dynamic rates, the current response range of the simulation software is not a limiting factor for testing flight algorithms and flight computer operation. If a different vehicle was to be simulated, an aircraft model for that vehicle would have to be developed and programmed into the simulation system.

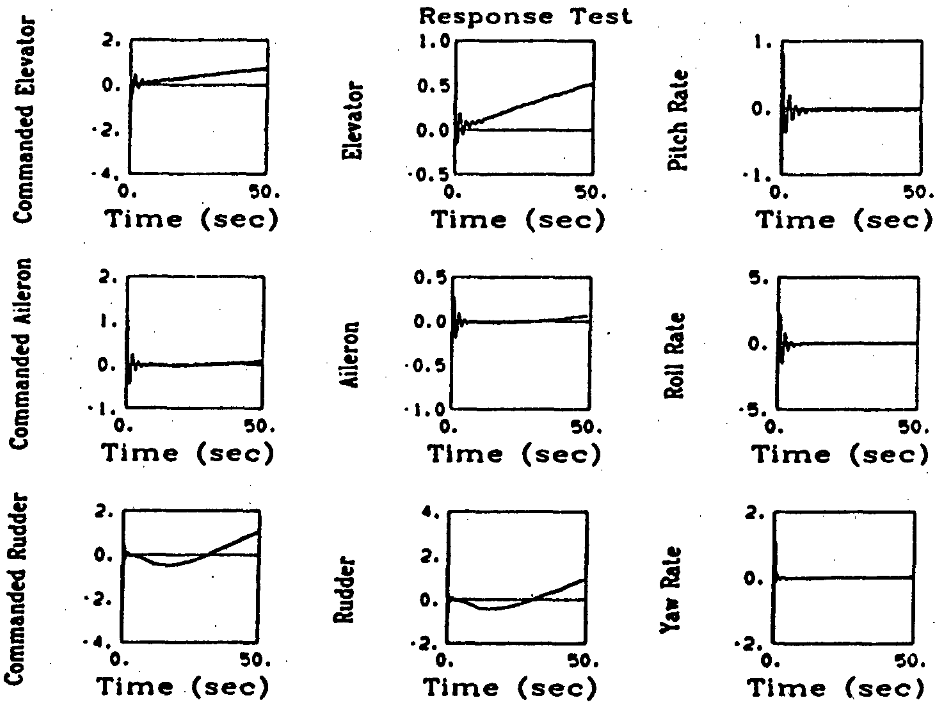


Figure 21: Plots of Simulated Aircraft Responses to Single Axis Inputs

The simulation system cannot replace flight tests. The model cannot accurately simulate unique flying conditions, such as losing or failing one or more control surfaces. These are some of the experiments that are ideal for testing on an unmanned aircraft since they are important to high performance, fighter aircraft, but are extremely dangerous to flight test on manned aircraft.

The current simulation system is designed to emulate the unmanned research vehicle and verify the operation of its flight computer, which runs at a 50 Hz update rate. Since it is non-intrusive to the flight computer, this simulation system can interface to any flight computer that controls this unmanned research vehicle and executes at 50 Hz. With minor modifications to the simulation software (for predicting the next state of the vehicle), a slower flight computer could also be verified. Due to the current execution rate of the simulation software, it would be more difficult to accurately verify the operation of a flight computer with a faster frame rate. The simulation system would either require a faster processor to increase its execution rate, or a simpler aircraft model to reduce execution time.

6.3 FUTURE ENHANCEMENTS

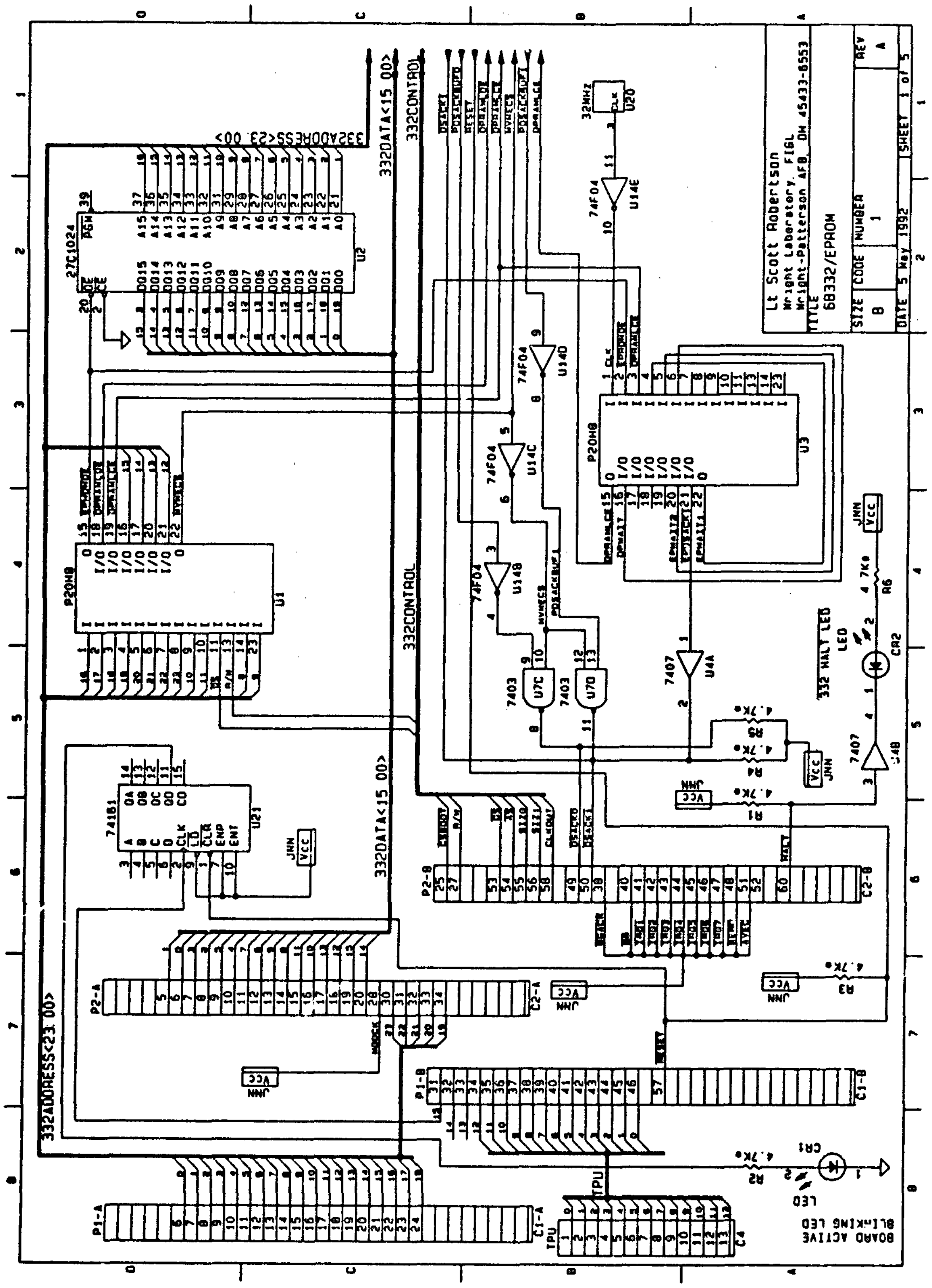
The 68030 processor board could be replaced with a more powerful processor board to either verify the operation of faster flight computers or execute more complex simulation software. While the current system can use the informational flight displays of Lambda's ground station, a more graphical front end which displays scenery could be developed to give the pilot a better feel of "flying" the flight computer.

6.4 CONCLUSIONS

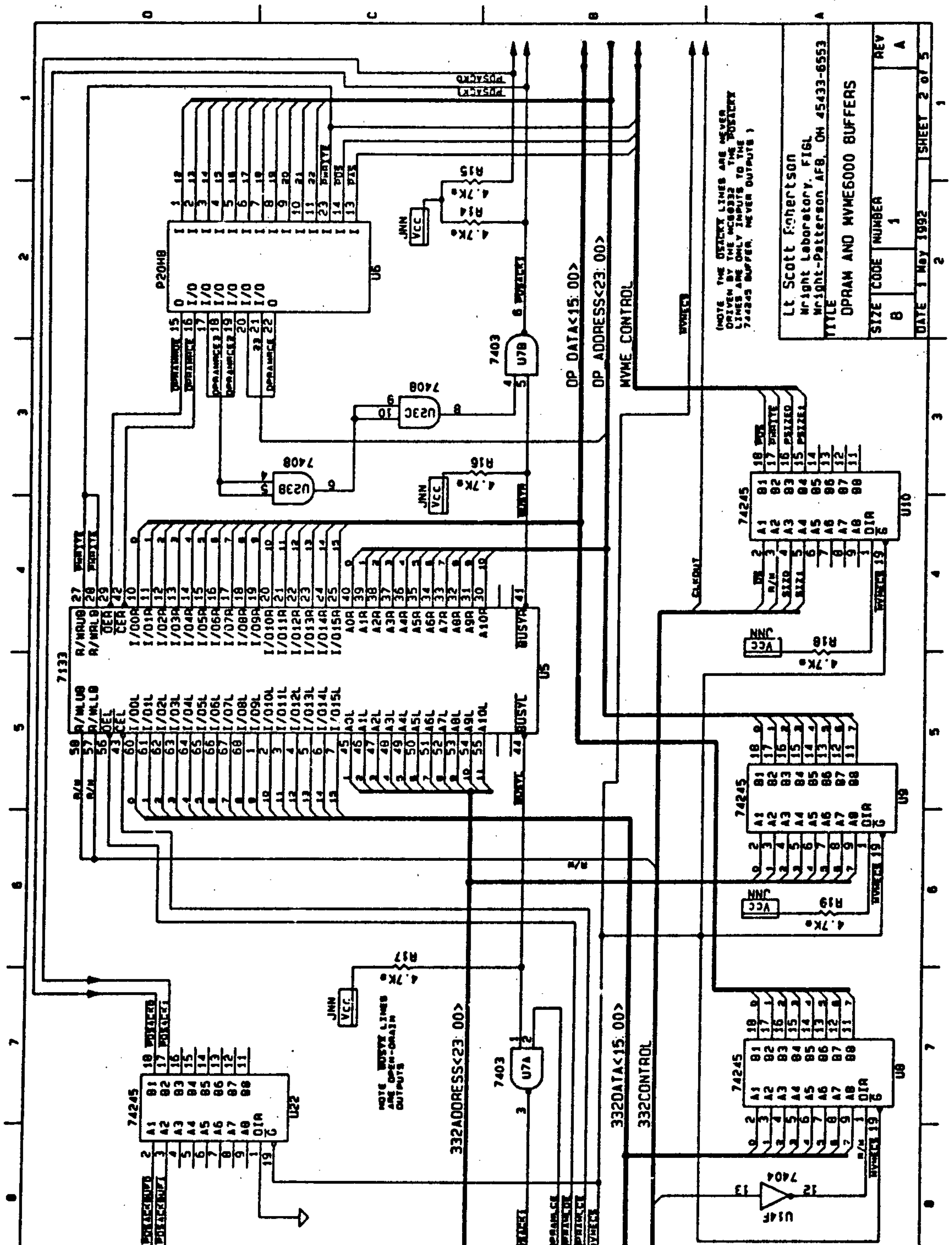
The simulation system meets all of its design objectives. It provides a real time, highly accurate simulation of an unmanned aerial research vehicle. The simulation system non-intrusively connects to the vehicle's flight computer, making the flight computer part of the simulation "loop". In this manner, the pilot can "fly" the flight computer, data can be collected, and the proper operation of the flight computer can be verified.

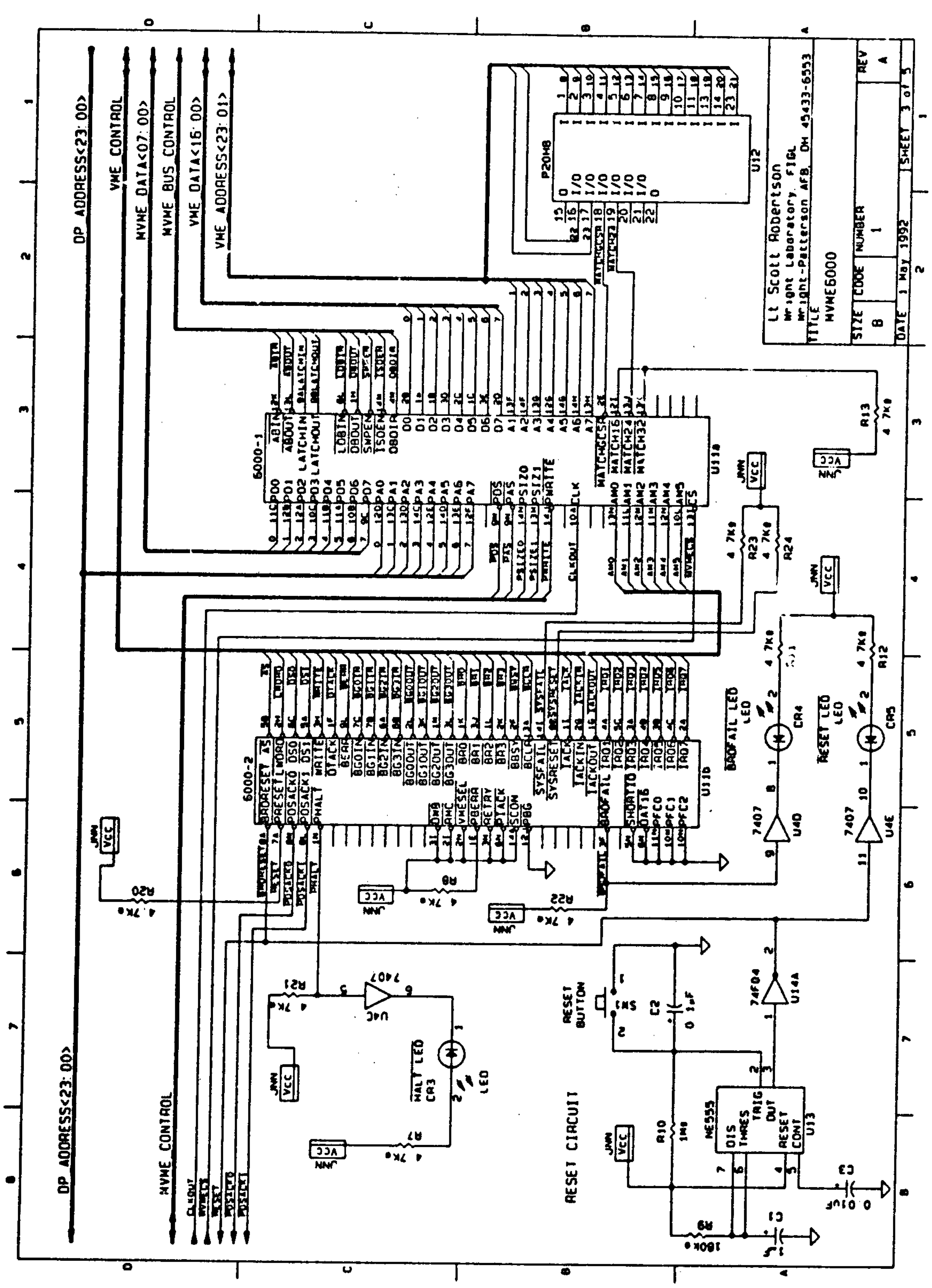
APPENDIX A

PULSE-WIDTH DEMODULATION BOARD SCHEMATICS

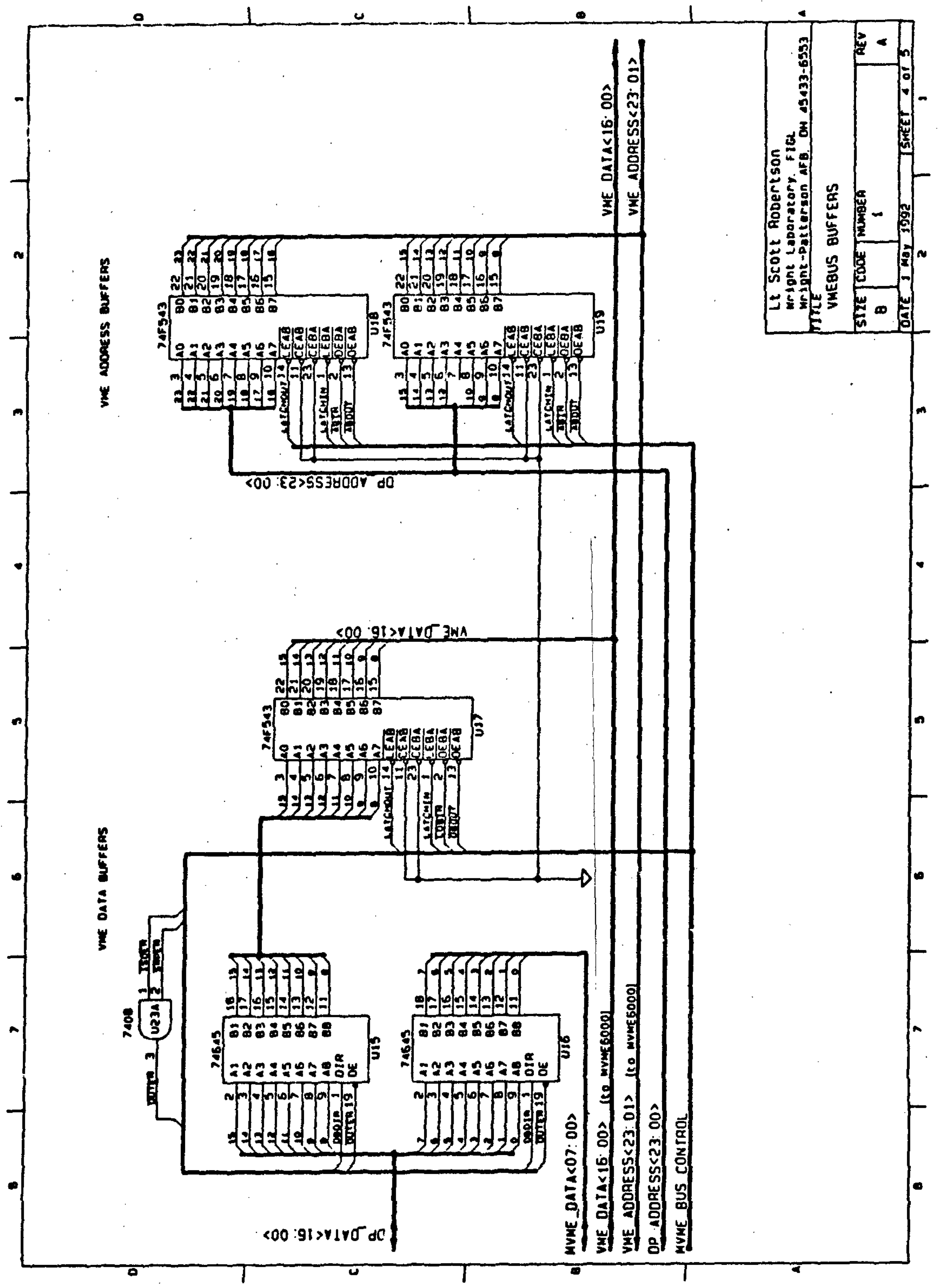


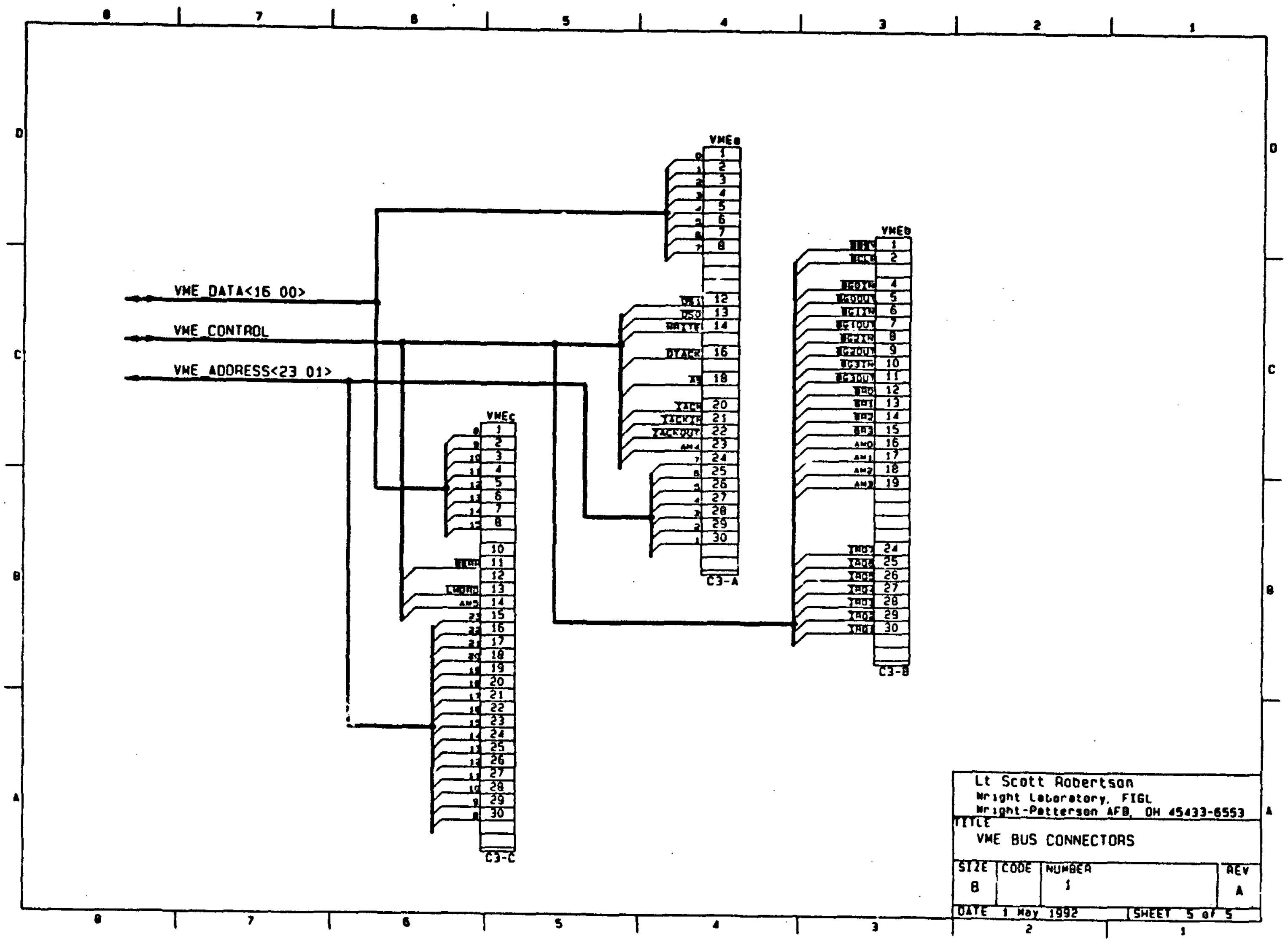
Lt Scott Robertson
Wright Laboratory, FIGL
Wright-Patterson AFB, OH 45433-6553
TITLE 68332/EPROM
SIZE B
CODE
NUMBER 1
REV A
DATE 5 May 1992
SHEET 1 of 5
332ADDRESS<23 00>
332DATA<15 00>
332CONTROL
27C1024
P20H8
74161
74F04
7403
7407
32MHz
332 HALT LED
BOARD ACTIVE BLINKING LED
TPU


Lt Scott Robertson
Wright Laboratory, FIGL
Wright-Patterson AFB, OH 45433-6553
TITLE DPRAM AND MVME6000 BUFFERS
SIZE B
CODE
NUMBER 1
REV A
DATE 1 May 1992
SHEET 2 of 5
7133
U5
P20H8
U6
74245
U8
U9
U10
U22
7403
U7A
U7B
7408
U23B
U23C
7404
U14F
R14
R15
R16
R17
R18
R19
4.7K
VCC
JNN
DP DATA<15:00>
DP ADDRESS<23:00>
MVME CONTROL
332ADDRESS<23:00>
332DATA<15:00>
332CONTROL
NOTE: BUSYL LINES ARE OPEN-DRAIN OUTPUTS
(NOTE: THE DSACKX LINES ARE NEVER DRIVEN BY THE MC68332. THE DSACKX LINES ARE ONLY INPUTS TO THE 74245 BUFFER, NEVER OUTPUTS.)



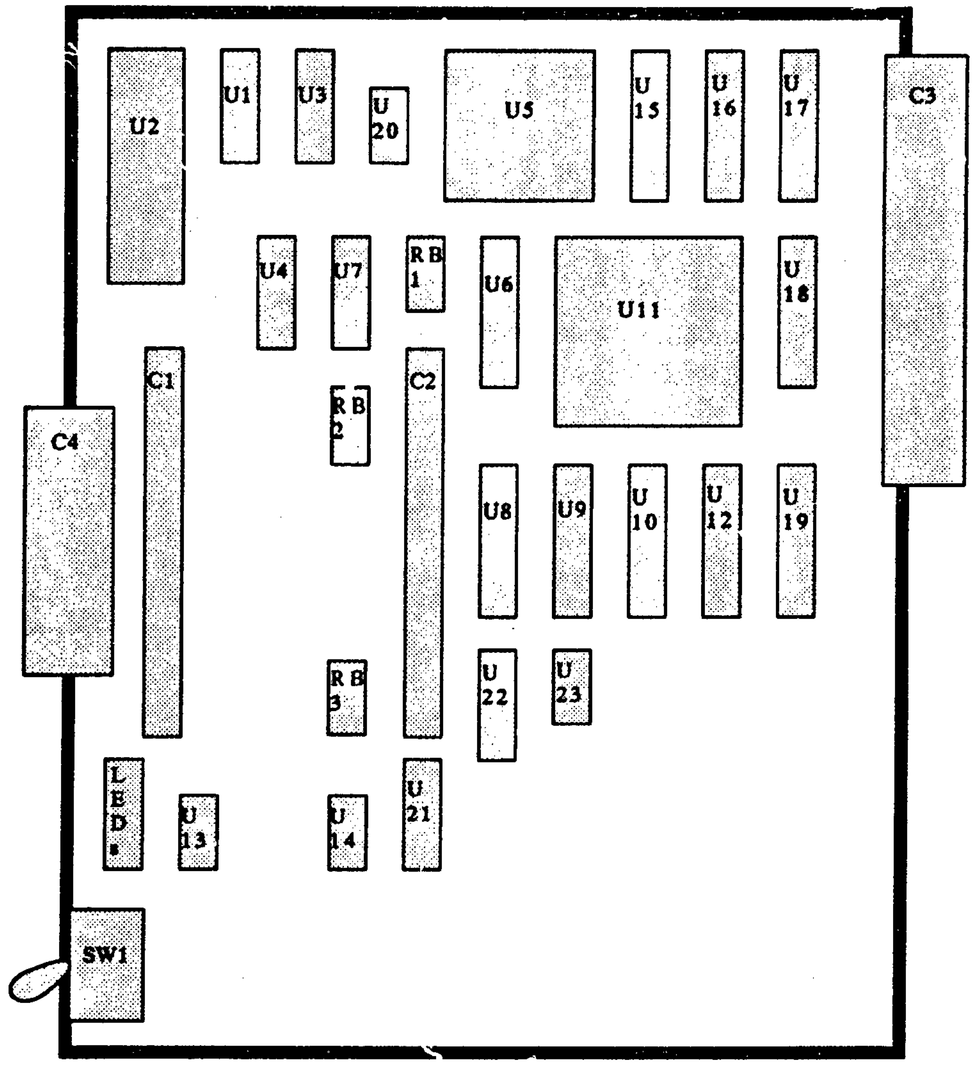
DP ADDRESS<23:00>
VME CONTROL
MVME DATA<07:00>
MVME BUS CONTROL
VME DATA<16:00>
VME ADDRESS<23:01>
MVME CONTROL
RESET CIRCUIT
6000-1
6000-2
U11a
U11b
U12
P20H8
U13
NE555
BADFAIL LED
RESET LED
HALT LED
RESET BUTTON
7407
74F04
Lt Scott Robertson
Wright Laboratory, FIGL
Wright-Patterson AFB, OH 45433-6553
TITLE
MVME6000
SIZE B
CODE
NUMBER 1
REV A
DATE 1 May 1992
SHEET 3 of 5


VME ADDRESS BUFFERS
VME DATA BUFFERS
74F543
U17
U18
U19
74645
U15
U16
7408
U23A
VME DATA<16:00>
VME ADDRESS<23:01>
DP ADDRESS<23:00>
DP_DATA<16:00>
MVME_DATA<07:00>
VME_DATA<16:00> (to MVME6000)
VME_ADDRESS<23:01> (to MVME6000)
DP_ADDRESS<23:00>
MVME_BUS_CONTROL
Lt Scott Robertson
Wright Laboratory, FIGL
Wright-Patterson AFB, OH 45433-6553
TITLE
VMEBUS BUFFERS
SIZE
B
CODE
NUMBER
1
REV
A
DATE 1 May 1992
SHEET 4 of 5


VME DATA<16 00>
VME CONTROL
VME ADDRESS<23 01>
VMEa
C3-A
VMEb
C3-B
VMEc
C3-C
Lt Scott Robertson
Wright Laboratory, FIGL
Wright-Patterson AFB, OH 45433-6553
TITLE
VME BUS CONNECTORS
SIZE B
CODE
NUMBER 1
REV A
DATE 1 May 1992
SHEET 5 of 5

APPENDIX B

PULSE-WIDTH DEMODULATION BOARD LAYOUT



Pulse-Width Demodulation Board Layout, Component Side

APPENDIX C

PULSE-WIDTH DEMODULATION BOARD PARTS LIST

PART NUMBER(s)	DEVICE	DESCRIPTION
U1	GAL20H8A	Programmable Generic Array Logic Device
U2	AM27C1024-150DC	1 Megabit (65,536 x 16-bit) CMOS EPROM
U3	GAL20H8A	Programmable Generic Array Logic Device
U4	SN7407	TTL Hex Buffer/Driver (with Open Collector High-Voltage Output)
U5	IDT7133S-55G	32K (2K x 16-Bit) CMOS Dual-Port RAM
U6	GAL20H8A	Programmable Generic Array Logic Device
U7	SN7403	TTL Quad 2-Input NAND Gate (with Open Collector High-Voltage Output)
U8, U9, U10	SN74F245N	Fast Octal Bus Transceiver (with 3-State Outputs)
U11	MVME6000AC	VMEbus Interface
U12	GAL20H8A	Programmable Generic Array Logic Device
U13	NE555	555 Timer
U14	SN74F04	Fast TTL Hex Inverter
U15, U16	SN74ALS645A-1	Octal Bus Transceivers (with 3-State Outputs)
U17, U18, U19	SN74F543	Fast Octal Registered Transceiver, Non-Inverting (with 3-State Outputs)

PART NUMBER(s)	DEVICE	DESCRIPTION
U20	Clock	32 MHz
U21	SN74F161A	Fast Binary Counter (Divide by 16)
U22	SN74F245N	Fast Octal Bus Transceiver (with 3-State Outputs)
U23	SN74F08	Fast TTL Quad 2-Input AND Gate
C1, C2	MC68332-BCC	Business Card Computer Connectors
C3	96-Pin, 3 Row Connector	VMEbus Connector
C4	20-Pin IDE Connector	Pulse-Width Signal Input Connector
RB1, RB2, RB3	Resistor Pack	15 220Ω Resistors
R9	180kΩ Resistor	5% fixed composition, ¼ Watt
R10	1MΩ	5% fixed composition, ¼ Watt
C1	1μf capacitor	tantalum
C2	0.1μf capacitor	tantalum
C3	0.01μf capacitor	ceramic disc
LED 1-5	red LED	
SW1	reset switch	

APPENDIX D

PROGRAMMABLE GENERIC ARRAY LOGIC DEVICE EQUATIONS

```
MODULE MC332
TITLE 'PERFORMS 332 ADDRESS DECODING FOR EPROM, DPRAM, MVME6000,
    AND 332 TO MVME6000 BUFFERS.  MVMECS DECODED TO EPROM
RANGE.
    SCOTT ROBERTSON 13 JUNE 1992';

    U1F                DEVICE 'P20V8S';

    A16, A17, A18, A19     PIN 1, 2, 3, 4;   " A19 = CS6
    A20, A21, A22, A23     PIN 5, 6, 7, 8;   " CS7, CS8, CS9, CS10
    A10, A11, _DS, RW      PIN 9, 10, 11, 13;
    A8, _EPROMOE           PIN 14, 15;
    _DPRAMLOE, _DPRAMLCE   PIN 18, 19;
    A15, A14, A13, A12     PIN 16, 17, 20, 21;
    _MVMECS, A9            PIN 22, 23;

    H, L, X             =1, 0, .X.;


EQUATIONS
    _EPROMOE = !(A23 & A22 & A21 & !A20 & !A19 &
          !A15 & !A14 & !_DS & RW); "$080000 - $083FFF

    _DPRAMLOE = !(A23 & A22 & !A21 & A20 & A19 & !A15 & !A14 &
          !A13 & !A12 & !_DS);  "$010XXX & READ

    _DPRAMLCE = !(((!A23 & !A22 & A21)#(A23 & A22 & !A21)) & A20
          & A19 & !A15 & !A14 & !A13 & !A12 & !_DS); "$010XXX

    _MVMECS = !(A23 & A22 & A21 & !A20 & !A19
          & !A15 & A14 & !A13 & A12 & !_DS); "$085XXX (& BUFFERS)

TEST_VECTORS
    ([ A23, A22, A21, A20, A19, A18, A17, A16, A15, A14, A13, A12, A11,
      A10, A9, A8, _DS, RW]
      -> [_EPROMOE, _DPRAMLOE, _DPRAMLCE, _MVMECS])
    [H, H, H, L, L, X, X, X, L, L, X, X, X, X, X, X, L, H]
     -> [L, H, H, H]; "_EPROMOE FOR $080000-083FFF

"    [L, L, L, L, L, L, L, L, L, L, L, L, L, L, L, L, L, H]
"     -> [L, H, H, H]; _EPROMOE FOR $0000XX

    [H, H, L, H, H, X, X, X, L, L, L, L, X, X, X, X, L, X]
     -> [H, L, L, H]; "_DP OE&CE FOR $010XXX AND R/W

    [L, L, H, H, H, X, X, X, L, L, L, L, X, X, X, X, L, X]
     -> [H, H, L, H]; "_DPRAMLCE FOR $010XXX AND !R/W

    [H, H, H, L, L, X, X, X, L, H, L, H, X, X, X, X, L, X]
     -> [H, H, H, L]; "_MVMECS FOR $085XXX

END MC332
```

```
MODULE DELAYGAL
TITLE 'PERFORMS EPROM DSACK1 DELAY FOR EPROM TO 332, AND ACTIVE
HIGH
    DPRAMLCE DELAY FOR NANDING WITH BUSYL SIGNAL FROM DPRAM
TO
    GENERATE DSACK1.
    SCOTT ROBERTSON 3 MAY 1992';

    U3              DEVICE 'P20V8R';  " 'R' - CLOCK IS USED.

    CLK             PIN 1;
    _EPROMOE            PIN 2;
    _DPRAMLCE            PIN 3;
    _EPWAIT1_IN         PIN 4;
    _EPWAIT2_IN         PIN 5;
    DPWAIT_IN          PIN 6;
    DPRAMLCE            PIN 15;
    DPWAIT_OUT           PIN 16;
    _EPWAIT2_OUT          PIN 20;
    _EPWAIT1_OUT          PIN 22;
    _EPDSACK1          PIN 21;

    H, L, X, C          =1, 0, .X., .C.;


EQUATIONS
    DPWAIT_OUT := !_DPRAMLCE;    "INVERTS SIGNAL (CLOCKED OUTPUT)
    DPRAMLCE := DPWAIT_IN;      " SIGNAL JUST DELAYED BY THE CLOCK
    !_EPWAIT1_OUT := !_EPROMOE;  "OUTPUT = INPUT, DLY'D BY THE CLK
    !_EPWAIT2_OUT := !_EPWAIT1_IN;  " SIGNAL JUST DELAYED BY THE
CLOCK
    !_EPDSACK1 := !_EPWAIT2_IN & !_EPROMOE; "DLY'D BY CLK AGAIN (2ND
TIME)
                        " _EPDSACK1 WILL BE NEGATED SHORTLY
                        " AFTER _EPROMOE IS NEGATED - NO DELAYS.

TEST_VECTORS
    ([CLK, _DPRAMLCE] -> [DPWAIT_OUT])

    [C, H] -> [L];
    [C, L] -> [H];

TEST_VECTORS
    ([CLK, DPWAIT_IN] -> [DPRAMLCE])

    [C, L] -> [L];
    [C, H] -> [H];
```

```
TEST_VECTORS
    ([CLK, _EPROMOE] -> [_EPWAIT1_OUT])

    [C, H] -> [H];
    [C, L] -> [L];

TEST_VECTORS
    ([CLK, _EPWAIT1_IN] -> [_EPWAIT2_OUT])

    [C, H] -> [H];
    [C, L] -> [L];

TEST_VECTORS
    ([CLK, _EPWAIT2_IN, _EPROMOE] -> [_EPDSACK1])

    [C, H, H] -> [H];
    [C, H, L] -> [H];
    [C, L, H] -> [H];
    [C, L, L] -> [L];

END DELAYGAL
```

```
MODULE DPRAMGAL
TITLE 'PERFORMS DPRAM ADDRESS DECODING FROM THE VME/MVME6000
SIDE.
     THE DPRAM WILL BE DECODED TO VME ADDRESS $40XXXX.
     SCOTT ROBERTSON, 3 MAY 1992';

     U6                DEVICE 'P20V8S';  "NO CLOCK USED

     A12, A13, A14, A15      PIN 1, 2, 3, 4;
     A16, A17, A18, A19      PIN 5, 6, 7, 8;
     A20, A21, A22, A23      PIN 9, 10, 11, 21;
     _PWRITE, _PDS, _PAS     PIN 23, 14, 13;
     _DPRAMROE, _DPRAMRCE    PIN 15, 16;
     DPRAMRCE0, DPRAMRCEIN   PIN 22, 20;
     DPRAMRCE1, DPRAMRCEIN2  PIN 19, 17;
     DPRAMRCE2               PIN 18;


     H, L, X              =1, 0, .X.;

EQUATIONS
     _DPRAMROE = !(!A23 & A22 & !A21 & !A20 & !A19 & !A18 & !A17 &
             !A16 & !_PAS & !_PDS & _PWRITE); "$40XXXX & READ

     _DPRAMRCE = !(!A23 & A22 & !A21 & !A20 & !A19 & !A18 & !A17 &
             !A16 & !_PAS & !_PDS); "$40XXXX

     DPRAMRCE0 = (!A23 & A22 & !A21 & !A20 & !A19 & !A18 & !A17 &
             !A16 & !_PAS & !_PDS); "$40XXXX, ACTIVE HIGH

     DPRAMRCE1 = DPRAMRCEIN;  "LOOP THE DPRAMCE0 BACK THROUGH
THE
                     " GAL FOR A DELAY IN ASSERTING PDSACK1.

     DPRAMRCE2 = DPRAMRCEIN2; "2ND DELAY LOOP.

TEST_VECTORS
     ([ A23, A22, A21, A20, A19, A18, A17, A16, _PAS, _PDS, _PWRITE]
      -> [_DPRAMROE, _DPRAMRCE, DPRAMRCE0])

     [L, H, L, L, L, L, L, L, L, L, H] -> [L, L, H]; "_DPRAMROE & CE'S
     [L, H, L, L, L, L, L, L, L, L, L] -> [H, L, H]; "_DPRAMRCE'S

TEST_VECTORS
     ([DPRAMRCEIN] -> [DPRAMRCE1])

     [L] -> [L];
     [H] -> [H];
```

```
TEST_VECTORS
    ([DPRAMRCEIN2] -> [DPRAMRCE2])

    [L] -> [L];
    [H] -> [H];

END DPRAMGAL
```

```
MODULE MVMEGAL
TITLE 'PERFORMS MVME6000 ADDRESS DECODING FROM THE VME BUS.
      THE MVME6000 WILL BE DECODED TO VME ADDRESS $400XXX.
      THE GCSR WILL BE DECODED TO $401XXX.
      SCOTT ROBERTSON, 3 MAY 1992';

      U12                DEVICE 'P20V8S';  " 'S' - NO CLOCK USED

      A8, A9, A10, A11        PIN 1, 2, 3, 4;
      A12, A13, A14, A15      PIN 5, 6, 7, 8;
      A16, A17, A18, A19      PIN 9, 10, 11, 13;
      A20, A21, A22, A23      PIN 14, 23, 16, 17;
      _MATCHGCSR, _MATCH24    PIN 18, 19;

      H, L, X              =1, 0, .X.;

EQUATIONS
     !_MATCHGCSR = (!A23 & A22 & !A21 & !A20 & !A19 & !A18 & !A17 &
                   !A16 & !A15 & !A14 & !A13 & A12); "$401XXX

     !_MATCH24 = (!A23 & A22 & !A21 & !A20 & !A19 & !A18 & !A17 &
                 !A16 & !A15 & !A14 & !A13 & !A12); "$400XXX

TEST_VECTORS
     ([ A23, A22, A21, A20, A19, A18, A17, A16, A15, A14, A13, A12]
      -> [_MATCHGCSR, _MATCH24])

     [L, H, L, L, L, L, L, L, L, L, L, H] -> [L, H];  "$401XXX
     [L, H, L, L, L, L, L, L, L, L, L, L] -> [H, L];  "$401XXX

END MVMEGAL
```

BIBLIOGRAPHY

Advanced Micro Devices. Memory Products Data Book. Advanced Micro Devices, Inc., 1989.

FutureNet. ABEL 3.0 User's Manual. FutureNet Division, Data I/O Corporation, 1988.

Clements, A. Microprocessor Systems Design - 68000 Hardware, Software, and Interfacing. PWS-Kent, 1987.

Cobalt Blue. FOR C: A FORTRAN to C Translator. Lightfoot & Associates, Inc., 1991.

Data I/O. FutureNet Schematic Designer User Manual. Data I/O Corp., 1991.

Fairchild. FAST, Fairchild Advanced Schottky TTL. Fairchild Camera and Instrument Corporation, 1985.

Fairchild. uALinear. Fairchild Camera and Instrument Corporation, 1982.

Fairchild. TTL Data Book. Fairchild Camera and Instrument Corporation, 1978.

Heurikon. HK68/V3D VMEbus 68030-based Single Board Computer User's Manual, Revision B. Heurikon Corp., 1991.

Integrated Device Technology. High Performance CMOS Data Book Supplement. Integrated Device Technology, Inc., 1989.

Lattice Semiconductor Corp. GAL DATA Book. Lattice Semiconductor Corp., 1988.

Motorola. CPU32 Central Processor Unit Reference Manual. Motorola, Inc., 1990.

Motorola. Linear and Interface Integrated Circuits. Motorola, Inc., 1985.

Motorola. M68300DIBUG Development Interface Debug Monitor User's Manual. Motorola, Inc., 1991.

Motorola. M68300 Family - Central Processing Unit Reference Manual. Motorola, Inc., 1990.

Motorola. M68300 Family - Time Processor Unit Reference Manual. Motorola, Inc., 1990.

Motorola. MC68030 User's Manual. Motorola, Inc., 1989.

Motorola. MC68332 System Integration Module User's Manual. Motorola, Inc., 1989.

Motorola. MC68332 User's Manual. Motorola, Inc., 1990.

Rafiquzzaman, M. Microprocessors and Microcomputer-based System Design. CRC Press, Inc., 1990.

Signetics. FAST Data Manual. Signetics Corp., 1987.

Software Development Systems. CrossCode C for the 68000 Microprocessor Family. Software Development Systems, Inc., 1990.

VMEbus International Trade Association. The VMEbus Specification ANSI/IEEE Standard 1014. VMEbus International Trade Association.

VME Microsystems International Corp. VMIVME-4132 32-Channel 12-Bit Analog Output Board with Built-In-Test. VME Microsystems International Corp., 1991.

Zilog. Z8036 C-CIO/Z8536 CIO Counter/Timer and Parallel I/O Unit Technical Manual. Zilog, Inc., 1987.

U.S. GOVERNMENT PRINTING OFFICE 750-113